Universidade de Aveiro
2016
Departamento de Economia, Gestão, Engenharia Industrial e Turismo

Sandra Isabel Ferreira de Sousa

Aproximações heurísticas para um problema de escalonamento do tipo flexible job-shop

Heuristic approaches for a flexible job-shop scheduling problem

Universidade de Aveiro
2016
Departamento de Economia, Gestão, Engenharia Industrial e Turismo

Sandra Isabel Ferreira de Sousa

Aproximações heurísticas para um problema de escalonamento do tipo flexible job-shop

Heuristic approaches for a flexible job-shop scheduling problem

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia e Gestão Industrial, realizada sob a orientação científica do Doutor Rui Jorge Ferreira Soares Borges Lopes, Professor Auxiliar do Departamento de Economia, Gestão, Engenharia Industrial e Turismo da Universidade de Aveiro.

Dedico este trabalho à minha família.

o júri

presidente

Prof. Doutor Carlos Manuel dos Santos Ferreira
professor associado c/ agregação da Universidade de Aveiro

Prof. Doutor Pedro Sanches Amorim
professor auxiliar da Faculdade de Engenharia da Universidade do Porto

Prof. Doutor Rui Jorge Ferreira Soares Borges Lopes
professor auxiliar da Universidade de Aveiro

agradecimentos

Ao meu orientador na Universidade de Aveiro, Professor Rui Borges Lopes, agradeço toda a disponibilidade demonstrada, o auxílio prestado e os sábios conselhos transmitidos durante a realização deste trabalho, assim como noutras ocasiões ao longo do meu percurso académico na Universidade de Aveiro.

Ao meu orientador na Softi9, Paulo Barreira, estou muito grata por me ter proporcionado a oportunidade de realizar este projecto de investigação na Softi9, e também por toda a disponibilidade e ensinamentos prestados.

Por fim, aos meus pais pelo apoio e motivação que me prestaram e por todos os outros meios providenciados que permitiram realizar a minha formação académica.

palavras-chave

Escalonamento, flexible job-shop, capacidade dos recursos, heurísticas, planeamento e escalonamento avançados.

resumo

Este trabalho aborda um novo tipo de problema de escalonamento que pode ser encontrado em várias aplicações do mundo-real, principalmente na indústria transformadora. Em relação à configuração do *shop floor*, o problema pode ser classificado como *flexible job-shop*, onde os trabalhos podem ter diferentes rotas ao longo dos recursos e as suas operações têm um conjunto de recursos onde podem ser realizadas. Outras características de processamento abordadas são: datas possíveis de início, restrições de precedência (entre operações de um mesmo trabalho ou entre diferentes trabalhos), capacidade dos recursos (incluindo paragens, alterações na capacidade e capacidade infinita) e tempos de *setup* (que podem ser dependentes ou independentes da sequência). O objetivo é minimizar o número total de trabalhos atrasados.

Para resolver o novo problema de escalonamento proposto um modelo de programação linear inteira mista é apresentado e novas abordagens heurísticas são propostas.

Duas heurísticas construtivas, cinco heurísticas de melhoramento e duas metaheurísticas são propostas. As heurísticas construtivas são baseadas em regras de ordenação simples, onde as principais diferenças entre elas dizem respeito às regras de ordenação utilizadas e à forma de atribuir os recursos às operações. Os métodos são designados de *job-by-job* (JBJ), *operation-by-operation* (OBO) e *resource-by-resource* (RBR). Dentro das heurísticas de melhoramento, a *reassign* e a *external exchange* visam alterar a atribuição dos recursos, a *internal exchange* e a *swap* pretendem alterar a sequência de operações e a *reinsert-reassign* é focada em mudar, simultaneamente, ambas as partes. Algumas das heurísticas propostas são usadas em metaheurísticas, nomeadamente a *greedy randomized adaptive search procedure* (GRASP) e a *iterated local search* (ILS).

Para avaliar estas abordagens, é proposto um novo conjunto de instâncias adaptadas de problemas de escalonamento gerais do tipo *flexible job-shop*. De todos os métodos, o que apresenta os melhores resultados é o ILS-OBO obtendo melhores valores médios de *gaps* em tempos médios inferiores a 3 minutos.

keywords

Scheduling, flexible job-shop, resources capacity, heuristics, advanced planning and scheduling.

abstract

This work addresses a new type of scheduling problem which can be found in several real-world applications, mostly in manufacturing. Regarding shop floor configuration, the problem can be classified as flexible job-shop, where jobs can have different routes passing through resources and their operations have a set of eligible resources in which they can be performed. The processing characteristics addressed are release dates, precedence constraints (either between operations of the same job or between different jobs), resources capacity (including downtimes, changes in capacity, and infinite capacity), and setup times, which can be sequence-dependent or sequence-independent. The objective is to minimise the total number of tardy jobs.

To tackle the newly proposed flexible job-shop scheduling problem (FJSP), a mixed integer linear programming model (MILP) is presented and new heuristic approaches are put forward.

Three constructive heuristics, five improvement heuristics, and two metaheuristics are proposed. The constructive heuristics are based on simple dispatching rules, where the main differences among them concern the used dispatching rules and the way resources are assigned. The methods are named job-by-job (JBJ), operation-by-operation (OBO) and resource-by-resource (RBR). Within improvement heuristics, reassign and external exchange aim to change the resources assignment, internal exchange and swap intend changing the operations sequence, and reinsert-reassign is focused in simultaneously changing both parts. Some of the proposed heuristics are used within metaheuristic frameworks, namely greedy randomized adaptive search procedure (GRASP) and iterative local search (ILS).

In order to evaluate these approaches, a new set of benchmark instances adapted from the general FJSP is proposed. Out of all methods, the one which shows the best average results is ILS-OBO obtaining the best average gap values in average times lower than 3 minutes.

Contents

List of Figures

List of Figures

List of Tables

List of Tables

List of Acronyms

ACO	Ant colony optimization
APS	Advanced planning and scheduling
B&B	Branch-and-bound
B&C	Branch-and-cut
B&P	Branch-and-price
CDR	Composite dispatching rules
EDD	Earliest due date
ERD	Earliest release date
FCFS	First come first served
FFSP	Flexible flow-shop scheduling problem
FISFS	First in system first served
FJSP	Flexible job-shop scheduling problem
FSP	Flow-shop scheduling problem
GA	Genetic algorithm
GRASP	Greedy randomized adaptive search procedure
HWC	Homogenous work centre
ILS	Iterated local search
JBJ	Job-by-job
JCS	Job-shop cell scheduling
JIT	Just-in-time
JSP	Job-shop scheduling problem
LPT	Longest processing time
LSF	Least slack first

List of Acronyms

LWR	Least work remaining
MILP	Mixed integer linear programming
MIP	Mixed integer programming
MOR	Most operations remaining
MWR	Most work remaining
NWC	Normal work centre
OBO	Operation-by-operation
OPT	Optimized production technology
OSP	Open-shop scheduling problem
PSO	Particle swarm optimization
RBR	Resource-by-resource
SA	Simulated annealing
SBP	Shifting bottleneck procedure
SDR	Simple dispatching rules
SMED	Single-minute exchange of die
SPT	Shortest processing time
TS	Tabu search
TSAFSP	Two-stage assembly flow-shop scheduling problem
VNS	Variable neighbourhood search

Chapter 1

Introduction

Optimization of resources is critical for the success of organizations. An efficient management of resources typically brings several advantages to organization's stakeholders: cost reduction, customer service improvement, increased productivity, business growth, among others. Although resource optimization is relevant to several types of organizations, it is in manufacturing industries, particularly in their production departments, that the application of optimization tools assumes greater importance. Production plays a key role in organizations' capacity to provide a prompt and effective response to customers' requests, and therefore in ensuring customer satisfaction. In order to achieve the objectives of production systems, all resources at the shop floor level should be synchronised in order to enable obtaining the desired products, in the required quantities and dates, and at the least possible cost.

Correct planning and scheduling is therefore essential. Planning ensures the availability of all that is needed for a given production horizon and scheduling intends to define the best sequence to produce customers' orders in the existing resources. As many of the resources are scarce or have finite capacity, and are subject to breakdowns, maintenance and setup activities (among other situations) a good scheduling solution is vital to meet most of the organizational goals for the shop floor (e.g. maximisation of resources utilization, fulfillment of delivery dates, setup minimisation).

In this context, Advanced Planning and Scheduling (APS) systems aim to aid decision-making in production management, more specifically, in tackling scheduling problems in a more efficient and effective way. Despite the increasing development of these decision support tools many industrial scheduling problems are still tackled manually. This may be due to the inability of optimization algorithms to deal with the underlying complexity of most real-world scheduling problems. Moreover, optimization algorithms often require a substantial investment which may not lead to meaningful improvements.

Scheduling problems have started being addressed in the scientific community since the 1950s. Ranging from simpler to more complex problems, many methods have been developed to solve these combinatorial optimization problems. The more complex is the production process, the higher the number of constraints imposed on the problem and the number of objectives to be optimized, making the scheduling problem harder to solve. In these cases heuristic approaches are typically the most suitable techniques to obtain good-quality solutions in reasonable time.

The present work arises from a project which was carried out at Softi9, a company in Aveiro, and had the duration of 7 months. One of the main business activities of Softi9 is the development and commercialization of the *Softinov APS* software. It is an information system directed at production planning and scheduling in industries with finite capacity. *Softinov APS* software is currently implemented in several types of industries: ceramic, special steels alloys, pharmaceutical, among others. Each of them has different scheduling problems, which are strongly related to specific characteristics of their production processes. Based on their most common features a new scheduling problem is presented. Some of the addressed characteristics are release dates, resources capacity, setup times and precedence constraints, and the objective is to minimise the total number of tardy jobs (i.e. production orders). Regarding shop floor configuration, this problem can be classified as a flexible job-shop scheduling problem (FJSP). As it is a NP-hard problem new heuristics approaches (constructive, improvements and metaheuristics) are proposed and validated.

1.1. Objectives

The main objectives to be achieved with this dissertation are the following:

- to review the current research on scheduling problems in the context of manufacturing environments, in order to identify most commonly studied scheduling problems as well as solution methods to tackle them.
- to address a scheduling problem, based on common manufacturing characteristics, potentially applicable to many real-world manufacturing industries.
- to develop new approaches to tackle the identified scheduling problem with the intent of increasing the knowledge on the problem and potentially improve decision support in scheduling problems.
- to test and validate the newly developed approaches to further understand their performance, and to provide benchmark instances for future studies of scheduling problems with similar characteristics.

1.2. Structure

This dissertation is composed of six chapters.

The first chapter introduces the scope of scheduling problems and briefly describes the scheduling problem addressed herein, the main objectives of this dissertation, and describes the structure of the dissertation.

The second chapter provides a literature review of scheduling problems in manufacturing environments. Based on the most commonly used classification in the literature, several variations of scheduling problems are reviewed and their practical applications identified. The most frequently used solution-finding approaches are briefly described, and some examples of recent works addressing them are provided.

In the third chapter a detailed description of the FJSP addressed in this dissertation and a mathematical formulation based on mixed-integer linear programming (MILP) is given.

As the addressed FJSP has not been studied in the literature before, the fourth chapter provides new constructive and improvements heuristics, and describes how two well-known metaheuristics (greedy randomized adaptive search procedure, GRASP, and iterated local search, ILS) were adapted to solve the problem.

A computational evaluation of all the aforementioned approaches is carried out in chapter 5. To test the approaches, several test instances were adapted from the FJSP literature and a real test instance obtained from a Softi9's customer. The performance of the approaches on the new test instances are reported and analysed, allowing to draw some preliminary conclusions.

Finally, in the last chapter, main conclusions and limitations of this work are presented and future work identified.

1. Introduction

Chapter 2

Scheduling Problems: a Literature Review

The basic scheduling problem aims finding the best order to process a given number of tasks, also called jobs, on a specified number of resources that are able to execute them (e.g. machines, people, facilities) over given time periods, such that one or more objectives is/are optimized (Hoogeveen, 2005; Pinedo, 2012).

The scheduling problem is often referred as sequencing, dispatching, or combinations thereof (Gupta and Stafford, 2006). However, sequencing is the process of defining the order in which jobs will be performed on resources whereas scheduling is the process of adding start and finish time information to the job order given by the sequence (Askin and Standridge, 1993). Therefore, scheduling problems encompass sequencing decisions, which typically must be carried out first.

Scheduling plays an important role in several real world environments in which scarce resources with often limited capacity have to be allocated to activities over time; a wide range of examples can be found in manufacturing, services, and information processing. Due to this, scheduling has received increasing attention in the literature, leading to the point where it is characterized by a virtually unlimited number of problem types (Brucker, 2007). Some of the most studied over the years are: project scheduling (e.g. Creemers, 2015; Liu et al., 2015); surgical operations scheduling (e.g. Riise et al., 2016; Wang et al., 2015); scheduling tasks in central processing units (e.g. Davidović and Crainic, 2015; Iturriaga et al., 2015); gate assignments at airports (e.g. Bouras et al., 2014; Pinedo, 2012); and courses or exams scheduling in universities (Burke et al., 2007; Czibula et al., 2016).

All these applications prove the extent and relevance of the subject; however, it is in manufacturing industries that scheduling activities gain particular importance. The survival of an organization depends on its ability to generate profit, requiring an appropriate cost management while meeting the highest customer demands. To achieve these goals organizations should constantly be looking to adopt policies that allow doing more and better while spending as little as possible. This is particularly felt at the organization's shop floor level where the main objective is to produce the right

goods subject to various constraints of the production process, always at the lowest possible cost and with the highest quality and celerity. To this end scheduling becomes a key factor for manufacturing productivity, as an effective scheduling can improve on-time delivery, reduce inventory, cut lead times, and improve the utilization of bottleneck resources (Tang and Liu, 2007; Wang et al., 1997). However reconciling the reach of all these goals is a very hard task making solving these problems often difficult, especially in a limited amount of time as is often the case in real-world applications.

In this chapter a literature review of scheduling problems in the context of manufacturing environments will be carried out. Firstly, it will be presented and explained the main classification for these problems, composed by three major parameters: shop floor configuration, processing characteristics and objective functions. Afterwards, typical solution methods are briefly introduced, which can be classified into exact or heuristic.

2.1. Classification of Scheduling Problems

Over the years several variants of scheduling problems have emerged in the literature, sometimes leading to authors proposing their own classifications. Out of the several existing classifications, in this work, the focus will be on the most broadly accepted.

According to Zobolas et al. (2008) one of the most widely used classification of scheduling problems is the shop floor configuration – number of resources, their arrangement and the flow pattern of jobs[1] among them. Based on this type of classification Graham et al. (1979) began to classify scheduling problems in five categories: single machine, parallel machines, flow-shop, job-shop and open-shop. Later Pinedo (2012) went further, proposing a classification which extend the parallel machine manufacturing environment to flow-shop and job-shop manufacturing environments. It is composed of seven main classes: single machine, parallel machines, flow-shop, flexible flow-shop, job-shop, flexible job-shop and open shop.

However, when classifying a specific scheduling problem often becomes important to consider other characteristics associated with the production process for a better identification/classification; some examples are job processing, setup requirements, and the performance measure to be optimized. With this aim two standard notations have emerged in the literature, including all of the previously mentioned characteristics, thus allowing classifying almost any type of deterministic scheduling problem.

[1] Where a given quantity of a specific product belongs to a production order.

The first of the standard notations was proposed by Conway et al. (1967) which was composed of four parameters: $\boldsymbol{A|B|C|D}$. For dynamic problems, $\boldsymbol{A}$ identifies the probability distribution of times between jobs arrivals. For static problems, $\boldsymbol{A}$ specifies the finite number of jobs, $\boldsymbol{B}$ identifies the number of resources in the shop floor, $\boldsymbol{C}$ describes the work-flow pattern in the shop floor, and $\boldsymbol{D}$ defines the criterion by which the schedule is evaluated.

The second standard notation is currently the most used in the literature (T'kindt and Billaut, 2006) and was introduced by Graham et al. (1979). Their classification is simpler than Conway et al.' one, being composed of just three fields: $\boldsymbol{\alpha}$, describing the shop floor environment; β, which concerns the process characteristics; and γ, for the chosen objective function(s). Therefore, each scheduling problem is described with a triplet $\boldsymbol{\alpha|\beta|\gamma}$.

Due to the virtually unlimited number of different production environments, over the time many variations of production scheduling problems have emerged. In order to cope with this evolution, the notations used for each of these three fields of Graham's notation were also updated. In Table 1 can be seen examples of currently used notations and corresponding descriptions for these three fields.

Table 1. Notation and description of α, β and γ fields according to the classification by Graham et al. (1979) (adapted from Allahverdi, 2015).

α		β		γ	
Notation	Description	Notation	Description	Notation	Description
1	Single machine	r_j	Non-zero release date/ ready time	C_{max}	Makespan
P or Pm	Parallel machines (identical)	$prmp$	Pre-emption	E_{max}	Maximum earliness
Q or Qm	Parallel machines (uniform)	$prec$	Precedence constraints	T_{max}	Maximum tardiness
R or Rm	Parallel machines (unrelated)	$fmls$	Job families	D_{max}	Maximum delivery time
Fm	m-stage flow-shop	$batch\ (b)$	Batch processing	W_{max}	Maximum resource workload
FFm	m-stage flexible (hybrid) flow-shop	ST_{sd}	Sequence-dependent setup time	TST	Total setup (changeover) time
AFm	m-stage assembly flow-shop	ST_{si}	Sequence-independent setup time	TNS	Total number of setups
J	Job-shop	$ST_{sd,f}$	Sequence-dependent family setup time	$\sum w_j C_j$	Total (weighted) completion time
FJ or FJm	Flexible Job-shop	$ST_{si,f}$	Sequence-independent family setup time	$\sum w_j F_j$	Total (weighted) flowtime
O	Open-shop	$prmu$ or $pmtn$	Permutation	$\sum w_j E_j$	Total (weighted) earliness
		$unavail$	Machine (un)availability	$\sum w_j T_j$	Total (weighted) tardiness
		M_j	Machine eligibility	$\sum w_j U_j$	Total (weighted) number of tardy jobs
		$block$	Blocking	$\sum w_k W_k$	Total (weighted) workload
		nwt	No-wait	$\sum w_j W_j$	Total (weighted) waiting time
		θ^{min}	Minimal time lag	$\sum h(E_j)$	Total earliness penalties
		θ^{max}	Maximal time lag	$\sum h(T_j)$	Total tardiness penalties
		$recrc$	Recirculation/Re-entrant	$\overline{F}$	Average (weighted) flow time
				$\overline{E}$	Average (weighted) earliness
				$\overline{T}$	Average (weighted) tardiness

A limitation of Graham's notation worth noting is that it only considers a single objective in the γ field. However, T'kindt and Billaut (2006) later extended it introducing a new parameterization of the γ field enabling classifying multi-objective problems.

In the following subsections each of the three parameters of the notation by Graham et al. (1979), adopted henceforth, will be explained in detail.

2.1.1. Shop Floor Configuration

For the adopted classification, parameter $\boldsymbol{\alpha}$ defines the configuration, i.e. the layout of the shop floor, including the number of stages (work centres) and routes of jobs through it. $\boldsymbol{\alpha}$ is sometimes composed by two sub-parameters: α_1, which concerns the general configuration of the shop (based on the routes of jobs among resources); α_2, which is the number of stages in the shop (Ruiz and Vázquez-Rodríguez, 2010; T'kindt and Billaut, 2006). Of the two sub-parameters, the most commonly used in scheduling problems' classification is α_1.

In Graham's classification (1979) the possible shop floor configurations, also called machine[2] environments, specified in the α field are single machine, parallel machines, flow-shop, job-shop and open-shop. Pinedo (2012) extended the number of shop floor configurations also considering flexible (hybrid) flow-shop and flexible job-shop. These are, respectively, a generalization of the flow-shop with parallel machine environments, and of the job-shop with parallel machine environments. Finally, Allahverdi (2015) also added the assembly flow-shop as a possible configuration.

The case of a **single machine** is the simplest of all possible machine environments (Pinedo, 2012): there is a single stage with a single resource to process a given number of jobs, each with a single operation (Allahverdi, 2015; Senthilkumar, 2010). According to Buffa and Sarin (1987) there are many situations where an entire plant can be viewed as a single machine, as is the case in chemical and paint manufacturing, and the manufacturing of products in automated plants. Another application is when solving more complex problems, which is often achieved by the study of single machine problems (Buffa and Sarin, 1987; T'kindt and Billaut, 2006). For instance, it is useful for solving more complex configurations where one resource is the bottleneck of the whole process and thus, generating a good schedule for the bottleneck resource is essential for the overall schedule

[2] According to Conway et al. (1967) the term *machine* concerns a resource capable of performing whatever has to be done in an operation (abstractly, it is a time scale with certain intervals of availability).

performance (Zobolas et al., 2008; Pinedo, 2009). Lately several applications of this problem have been investigated. Some examples can be seen in Herr and Goel (2016) and Chatavithee et al. (2015).

The **parallel machines** environment is a generalization of the single machine case, where each job has a single operation which can be performed by any resource of a set of resources working in parallel at a single stage (Allahverdi, 2015; Pinedo, 2009). Gholami and Sotskov (2014) note that using a set of parallel resources often allows increasing the throughput rate and avoids line stoppage when a resource fail occurs.

Resources in parallel may either be identical, uniform, or unrelated (Potts and Kovalyov, 2000). According to Gordon et al. (2002) and Senthilkumar (2010) these three types of resources can be defined as follows. Identical resources operate at the same speed for all jobs. Uniform resources, also called proportional or related, operate at different speeds which are independent of the jobs. Unrelated resources have a job-dependent speed.

Parallel machine scheduling problems share the same practical applications of single machine scheduling problems. The only difference is in the number of resources for processing the jobs. Li et al. (2015) studied scheduling identical parallel batch processing machines integrating production and delivery, while Chuang et al. (2010) addressed parallel machine scheduling in aluminium foil production.

Over the years characteristics of parallel machine environment have been extended to other manufacturing environments such as flow-shop and job-shop. This has led to the appearance of hybrid shop floor configurations, namely flexible flow-shop and flexible job-shop (Gholami and Sotskov, 2014; Ruiz and Vázquez-Rodríguez, 2010).

A **flow-shop** manufacturing environment (Figure 1a) is a special case of job-shop environment (Buffa and Sarin, 1987). It is often related to production of large quantities of a small diversity of products and is characterized by more or less continuous and unidirectional workflow of jobs through two or more stages in series, each stage having only one resource (Gupta and Stafford, 2006). If at least one stage has two or more parallel resources then the shop floor configuration is called **flexible (or hybrid) flow-shop** (Figure 1b). The duplication of the number of resources in some stages allows introducing flexibility, increase capacity and avoiding bottlenecks.

Basically flow-shop scheduling problem (FSP) consists in finding a sequence for processing a set of jobs in a set of stages so that one or more objectives is/are optimized. The particularity of such problem is that all jobs have the same sequence of operations across the stages, i.e. each job have to be processed first in stage 1 then on stage 2 and so on until last stage (Pan and Ruiz, 2013).

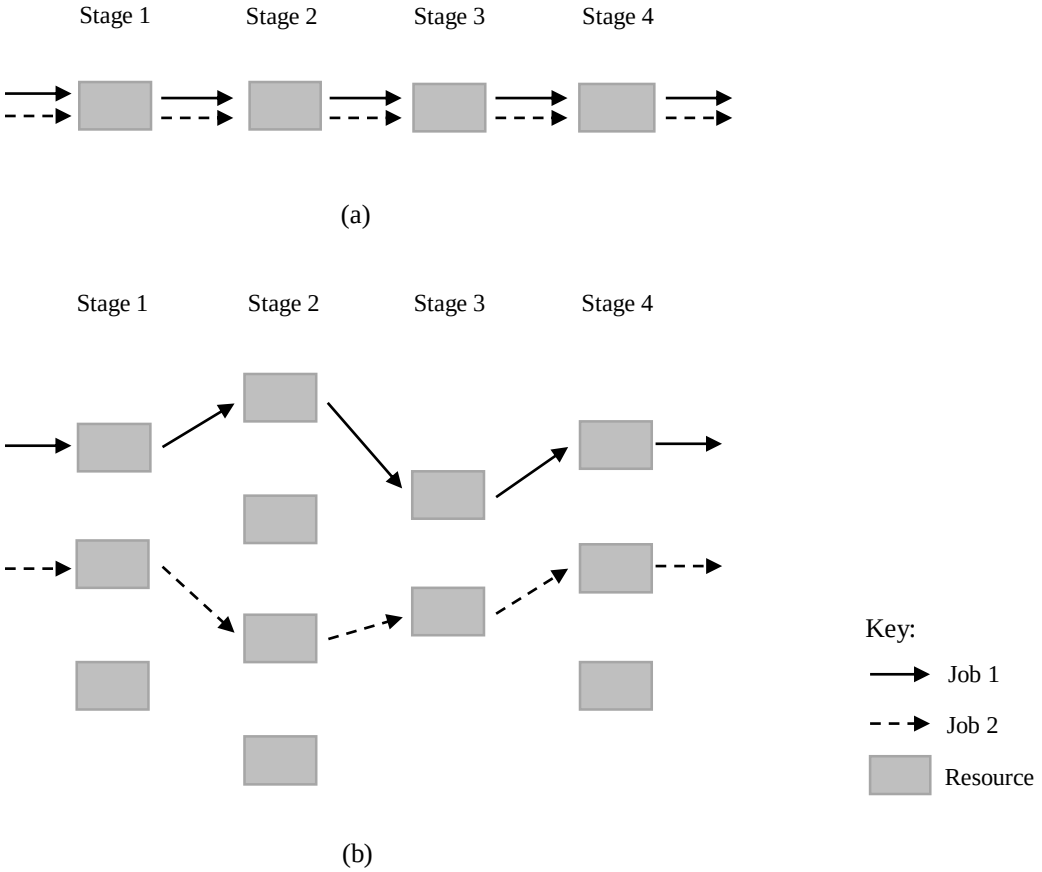

Figure 1. Flow-shop (a) and flexible flow-shop (b) environments.

Through Johnson's work published in 1954, it can be said that the FSP was the first class of scheduling problems being investigated. Since then many researchers have studied it due to its importance to manufacturing systems (Fernandez-viagas and Framinan, 2015; Jeong and Kim, 2014). The flexible flow-shop scheduling problem (FFSP) emerged afterwards (Gupta and Stafford, 2006). According to Liao et al. (2012) it is more commonly seen in industries such as glass, steel, paper and textile. Due to its practical relevance, FFSP has attracted significant attention from both researchers and practitioners (Lei and Guo, 2016; Nejati et al., 2014).

A special case of flow-shop environment which has received less attention in the literature, although commonly found in the real-world is the **assembly flow-shop** (see Figure 2). It is seen as a hybrid production system in which different parts are manufactured independently on parallel lines, and then the final product is produced by assembling these parts together (Shoaardebili and Fattahi, 2015). The underlying scheduling problem is known as the two-stage assembly flow-shop scheduling problem – TSAFSP – which according to Potts et al. (1995) is frequently found in practice, even more are as industries increasingly move to Just-In-Time (JIT) systems. An example provided by Potts et al. (1995) is the production of personal computers where a job is composed of a specific set

of modules (e.g. a central processing unit, a hard disc, a monitor, a keyboard, etc.) which are produced on independent lines and then are assembled according to customer specifications at an assembly stage. Recent studies about the TSAFSP can be found in the works of Allahverdi and Aydilek (2015) and Navaei et al. (2014).

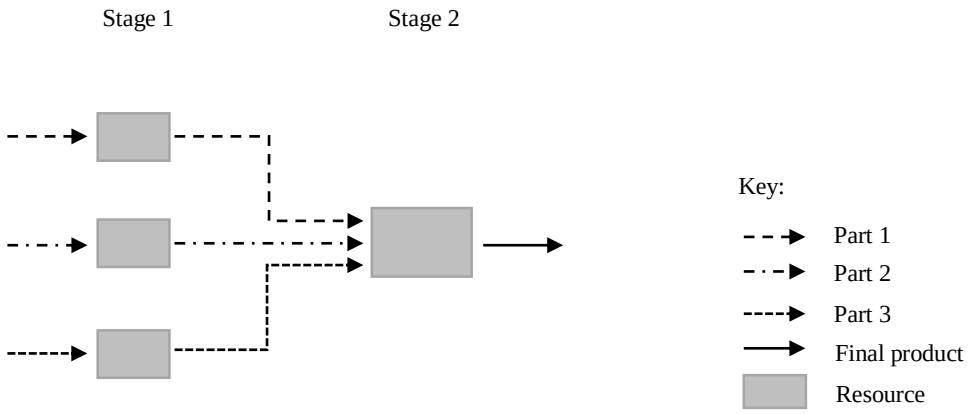

Figure 2. Assembly flow-shop environment.

Currently many industrial plants handle manufacturing of small quantities of a large diversity of products. This makes that each job often has its own sequence of operations (e.g. wafer fabrication in the semiconductor industry) thus the workflow does not follow a particular pattern (Peng et al., 2015). These characteristics can be found in **job-shop** environments (Figure 3a), also referred as a generalization of flow-shop (Pinedo, 2009). Similar to these latter cases, there may exist more than one resource in at least one stage, which is called a **flexible job-shop** environment (Figure 3b) (Allahverdi, 2015). Whereas in job-shops each job's operation requires the exclusive use of exactly one resource, in flexible job-shops it has a set of alternative resources where it can be performed, possibly with different processing times. This makes FJSP even more difficult to solve than the job-shop scheduling problem (JSP) which is considered one of the most difficult scheduling problems (Demir and İşleyen, 2014; Pezzella et al., 2008). This is due to the fact that in JSPs each job's route throughout the shop floor is fixed, while in FJSPs it becomes a decision variable concerning the assignment of each job's operation to one of its eligible resources. Thus, FJSPs are composed of two sub-problems: the assignment sub-problem and the sequencing sub-problem (Fattahi et al., 2009; Rossi, 2014). The latter, sequencing, consists in ordering the assigned operations on all resources for obtaining a feasible schedule.

According to Kacem et al. (2002) FJSP can be further divided into two categories: T-FJSP and P-FJSP. In T-FJSP, each operation can be processed in all resources of the shop floor (as each resource

is able to perform more than one type of operation) while in P-FJSP, at least one operation cannot be processed in all the resources (Gao et al., 2014).

JSP and FJSP are possibly the most similar to real-world production scenarios (Gao et al., 2008; Wang et al., 2010). Several practical applications for these problems can be found: printing and boarding industry (Vilcot and Billaut, 2011); manufacturing of integrated circuits on silicon wafers (Mason et al., 2002); glass industry (Alvarez-Valdes et al., 2005); mould manufacturing (Caballero-Villalobos et al., 2013) and other industrial sectors such as mechanical manufacturing and automobile assembly process (Gao et al., 2015). This may justify the increase of popularity of both problems in recent years (Gao et al., 2014).

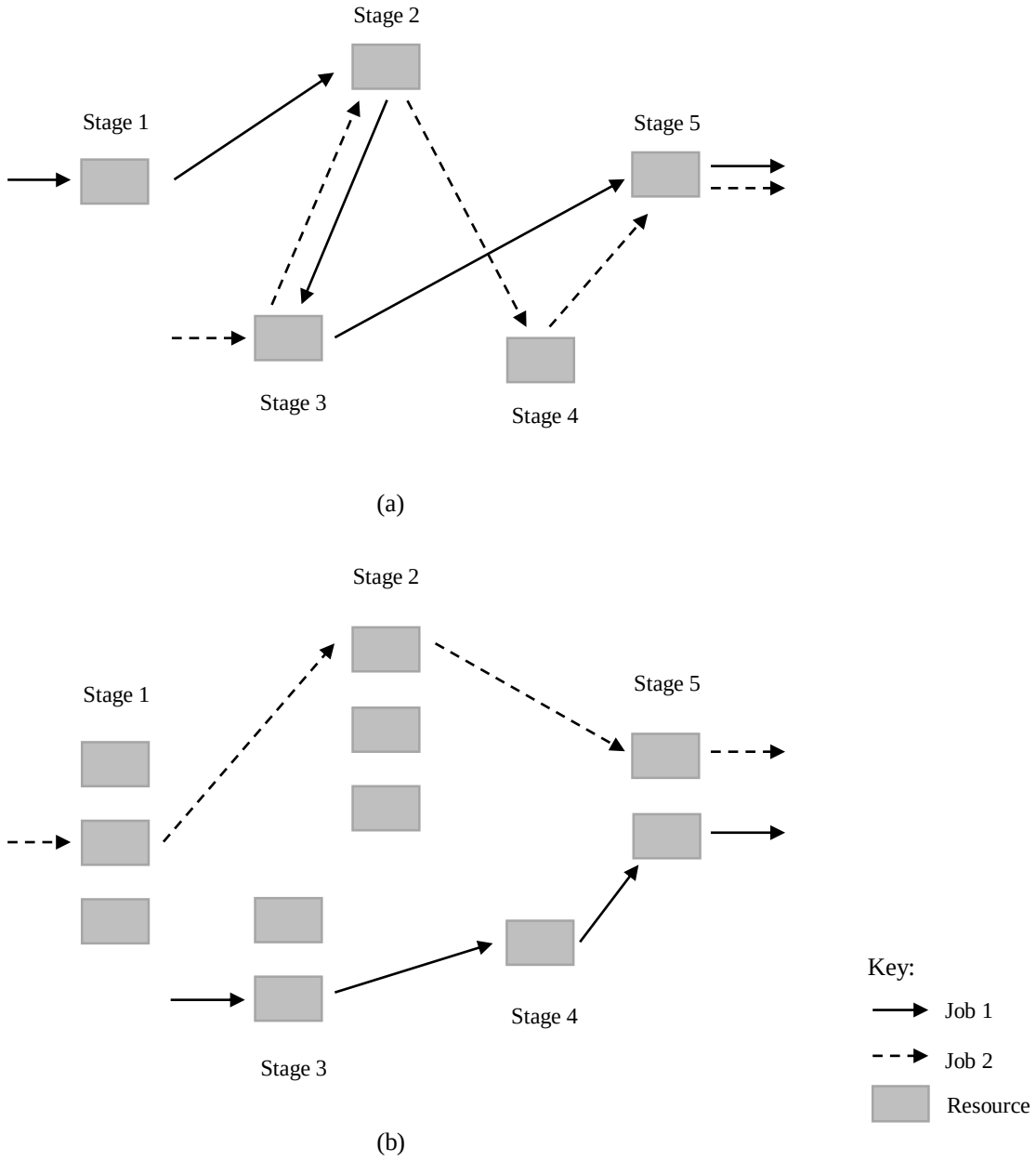

Figure 3. Job-shop (a) and flexible job-shop (b) environments.

In the previously mentioned shop floor environments each job has a predetermined fixed operations sequence; in the absence of this sequence it can be performed in any order. This is called **open-shop** environment (Allahverdi, 2015; Pinedo, 2012). Zobolas et al. (2008) suggests the car repair shop as the best example of an open shop environment, where the operation/repair sequence is not strictly define. Compared to other scheduling problems, the open-shop scheduling problem (OSP) has been rarely addressed in the literature. Works addressing the OSP are by Azadeh et al. (2015) and Panwalkar and Koulamas (2014).

2.1.2. Processing Characteristics

The second parameter of Graham's classification of scheduling problems is $\boldsymbol{\beta}$ which may contain a single entry, multiple entries, or no entry at all. It lists the constraints of a given problem due to its own particularities, often related to job characteristics, production processes characteristics, and other shop floor conditions (Allahverdi, 2015; Pinedo, 2012). When combined with shop floor configuration (α) it allows creating different variants of scheduling problems and consequently modelling manufacturing environments more realistically.

In manufacturing, a job is considered a production order, which has several characteristics. One of its characteristics is the **release date/ready time,** indicating when a given job can start its processing. This may depends on several factors (most noticeable one is the availability of materials and resources in order to process it) and determines the scheduling start of a given job. If the release date/ready time appears in β, jobs processing cannot start before that instant (Pinedo, 2012), also known as non-zero release date.

An important processing requirement which must always hold is **precedence constraints**; when existing, they can be between operations from the same job and/or between operations from different jobs. According to Lee et al. (2012) precedence constraints are divided into 'AND' and 'OR' types. In the 'AND' case an operation can be started only after all of its immediate predecessors have been completed. The 'OR' precedence constraint allows operations to start once one of its immediate predecessors has been completed. Figure 4 shows a network and a Gantt chart of several operations from a job. In the figure, operation O_5 can start after one of its immediate predecessors (O_3 and O_4) has been completed, i.e. an 'OR' precedence constraint; the remaining are 'AND' precedence constraints.

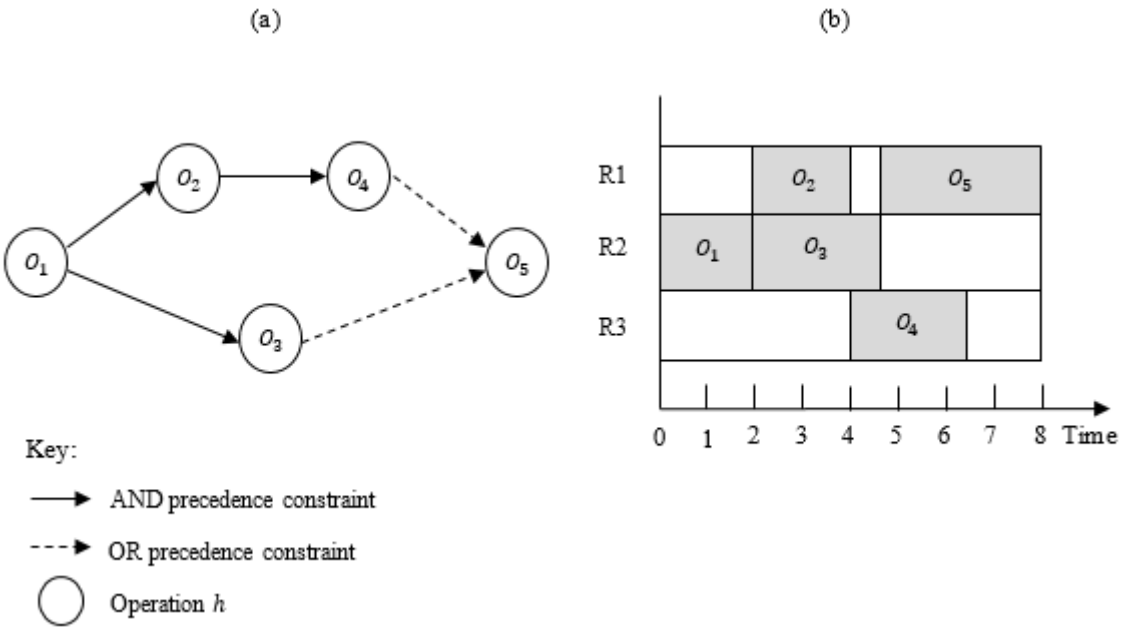

Figure 4. An example of (a) a network containing 'AND' and 'OR' precedence constraints and (b) the corresponding Gantt chart.

Another constraint commonly used in scheduling problems is **pre-emption**. It implies that it is not necessary to keep a job on a resource from start to finish, i.e. the processing of a job can be interrupted for example for making the resource available for a higher priority order (Pinedo, 2009). Most scheduling problems in the literature still do not allow pre-emption; some works addressing it are, for example, Seidgar et al. (2015) and Zhang and Yang (2016).

Currently, many manufacturing plants work with **job families** and **batch processing.** When several jobs have similar characteristics (e.g. operation sequence, tooling and setups) they can be grouped into a job family. A batch is a set of jobs of the same family. Thus, a batch processing resource can process several jobs simultaneously as long as its capacity is not exceeded (Allahverdi, 2015; Li et al., 2012). According to Lei and Guo (2011) some practical applications of batch processing resources are heat-treating ovens, chemical processes performed in tanks or kilns, testing process of electrical circuits and wafer fabrication process.

When a resource has just finished processing a job of a given family and has to start processing a job from another family it often requires a **setup (changeover) time**. Trietsch (1992), following Shingo (1985), defines setup time as the time elapsed from the moment a resource finishes one job until it starts working on the next job producing quality items. The setup time is often considered waste because no added value is provided during this time, and thus it should be reduced to the lowest

possible value (Shingo, 1985). Its reduction usually leads to significant increases of efficiency and is the focus of the Single-Minute Exchange of Die (SMED) methodology (Shingo, 1985). Regarding scheduling problems, setup times have been often overlooked. According to Allahverdi (2015) this may be valid for some scheduling applications where setup times are too short (e.g. automated manufacturing systems) but it adversely affects the solution quality (validity) of some other applications where setup times are longer (e.g. printing, textile, pharmaceutical, electronic, food processing, container/bottle industries).

Due to the importance of separating setup times from processing times, Allahverdi et al. (2008) propose a classification for scheduling problems taking into consideration setup times (Figure 5).

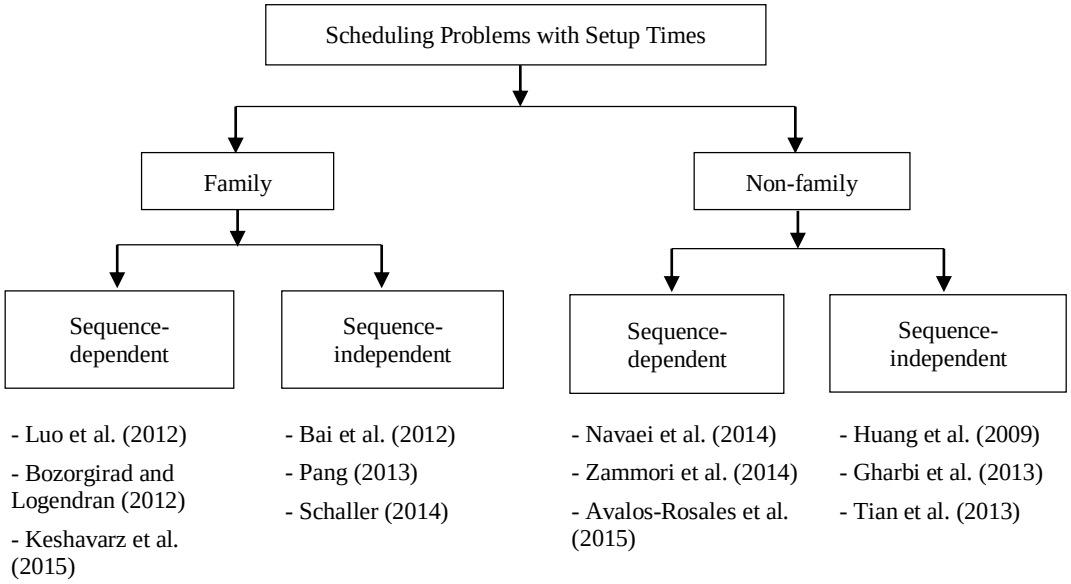

Figure 5. Classification of setup times in scheduling problems (adapted from Allahverdi et al., 2008) with some works addressing them.

When a scheduling problem involves several job families, changeovers on resources may be required. Setup times for changing from one job to another of the same family are often negligible, being larger and more impactful when changing between jobs of different families. This lead Allahverdi (2015) to classify setup times into two groups, family or non-family, further dividing these groups into sequence-dependent or sequence-independent. Setup times are called sequence-independent when depending solely on the task to be processed, regardless of its preceding task; sequence-dependent is when it depends on both the task and its preceding task.

As previously mentioned, in a flow-shop environment each job has the same sequence of operations through the stages. If we add the constraint that no job can pass another in the following resource (i.e. there is no *job passing*), the processing order of jobs is the same for all resources. This is known as **permutation** FSP and is associated with many types of industries such as, among others, chemical, petrochemical, automobile manufacturing, metallurgical and textile (Fernandez-viagas and Framinan, 2015; Molina-sánchez and González-Neira, 2016). On the other hand, if *job passing* is allowed the problem is called non-permutation FSP (Benavides and Ritt, 2016; Pan and Ruiz, 2013).

Specifically focusing on resources, several constraints may be envisioned. For example, unavailability due to shifts, scheduled maintenance and/or holidays, causing resources to not be continuously available, referred to as **machine (un)availability** or breakdowns (Pinedo, 2012; T'kindt and Billaut, 2006). Another example is when some resources are not able to perform a given job due to its special characteristics, making the job restricted to a limited set of resources, leading to **machine eligibility** constraints (Costa et al., 2014; Tadayon and Salmasi, 2012).

Another often used constraint is **blocking**, which implies that buffer capacities between two stages are limited or storage is not allowed in some stages (e.g. due to technological requirements), making that jobs must wait in the previous stage until sufficient space is released at the next stage (Grabowski and Pempera, 2007; T'kindt and Billaut, 2006). Examples of works addressing this constraint are Zhang and Gu (2015) and Abdollahpour and Rezaeian (2015) where FSP with limited intermediate buffers are considered. Some applications can be found in (Grabowski and Pempera, 2000; Ronconi, 2004; Zhang and Gu, 2015): chemical industry, manufacturing of concrete blocks, steel industry, manufacturing cells, production of electronic products, biological products and pharmaceuticals.

On the other hand, in several manufacturing processes jobs are not allowed to wait between two successive operations, i.e. the succeeding operation starts immediately after the preceding operation is completed. Jolai et al. (2009) mentioned that such situation occurs, for example, in some chemical and petrochemical processing, plastic moulding and silverware productions, in which a series of processes must follow one another immediately to prevent degrading. This requirement is named **no-wait** (Panwalkar and Koulamas, 2014; Pinedo, 2012). Another reason often pointed for no-wait between operations is the lack of storage between intermediate resources that industries often face (Ramezani et al., 2015).

However in some production environments there needs to be waiting time intervals between successive operations of the same job (Dhouib et al., 2013). These intervals have a lower and upper bounds which are respectively called minimal and maximal **time lags** (Fondrevelle et al., 2008). This can be found in several situations, such as food preparation prior to canning and in the drying of the glaze on ceramic tile manufacturing before kiln firing (Ruiz et al., 2008).

Flexible production systems are often composed of resources with the ability of performing various types of operations. One example is the flexible job-shop environment described in Section 2.1.1.. In these conditions jobs are allowed to be processed more than once in the same stage, i.e. can be **recirculation/re-entrant** of that jobs in the manufacturing system (T'kindt and Billaut, 2006). Other examples are production processes requiring quality testing and repairing, e.g. Jeong and Kim (2014) and Shin (2015) used re-entrant constraint in semiconductor manufacturing systems.

Other relevant characteristics found in scheduling problems are: transportation times of jobs between resources (Saidi-Mehrabad et al., 2015; Zabihzadeh and Rezaeian, 2016); overlapping in operations/lot-streaming, which consists in splitting production lots into smaller sublots such that each of them is treated individually and transferred to the next stage upon its completion (Han et al., 2016; Nejati et al., 2014); and individual operation's rejection/job rejection where some operations of a job (or a job) can either be produced by the company itself or purchased from suppliers, i.e. rejected (Gao and Lu, 2014; Neto and Filho, 2011). Although these are relevant processing characteristics, there is still no notation to identify them in the β field.

2.1.3. Objective Functions

The third parameter of Graham's classification of scheduling problems is $\boldsymbol{\gamma}$, which describes the objective to be minimised or maximised in order to evaluate solution (schedule) quality. It could contain a single entry (Pinedo, 2012; Graham, 1979) or multiple entries (T'kindt and Billaut, 2006), each entry corresponding to an objective function.

According to Pinedo (2012) and Lawler et al. (1993) objective functions can be of two types: regular or non-regular. Regular objective functions are non-decreasing in each of the jobs completion time $(C_1, \dots, C_j)$ and are often related to jobs completion time (e.g. makespan and total weighted[3] completion time) as well as its tardiness (e.g. maximum tardiness, total weighted tardiness, and total weighted number of tardy jobs). On the other hand objective functions considered non-regular are non-increasing and are related to the earliness of jobs (e.g. maximum earliness and total weighted earliness), and to the total number of setups.

Apart from this classification Graham et al. (1979) also group objective functions into: minimisation of a maximum measure, such as makespan $(C_{\max})$, tardiness (T_{max}), earliness (E_{max}) and minimisation of a total measure (e.g. time or number of jobs: $\sum w_j C_j, \sum w_j T_j, \sum w_j U_j$).

[3] Objective functions with weights allow considering jobs with different priorities.

According to Ruiz and Stützle (2008) minimisation of **makespan** (C_{max}) is one of the most commonly used objectives in the literature. It corresponds to the completion time of the last job (i.e. the time needed to finish all jobs) and evaluates schedules in terms of resources utilization (Benavides and Ritt, 2016; Fernandez-viagas and Framinan, 2015). However, in real manufacturing environments often the most favoured performance measures of schedulers are related with the ability to comply with jobs' due dates. Strict compliance with due dates of customers' orders, ensuring their satisfaction and at the same time avoiding stock costs is a goal of most companies. Therefore, performance measures related with due date of jobs has received increasing attention in recent years, including: minimisation of **total (weighted) tardiness** (Braune and Zäpfel, 2016; Jeong and Kim, 2014), **average (weighted) tardiness** (Calleja and Pastor, 2014), **total (weighted) number of tardy jobs** (Dhouib et al., 2013; Rodrigues et al., 2014), **maximum tardiness** (Akkan, 2015; Süer et al., 2014), **total (weighted) earliness, average (weighted) earliness** and **maximum earliness** (Koulamas and Panwalkar, 2015)**.**

Kuhpfahl and Bierwirth (2016) define the tardiness of a job (T_j) as the positive difference between its completion time (C_j) and due date (d_j). Hence, a job is tardy if its processing is completed after its due date; otherwise, it is considered an early job (Varmazyar and Salmasi, 2012).

Focusing on a single objective function, however, is often not realistic for many businesses. According to Wang et al. (2010), different departments in an enterprise have different expectations in order to maximise their own interests. The manufacturing department expects to reduce costs and improve work efficiency, corporate executives want to maximise the utilization of existing resources, and sales department hopes to better meet the delivery requirements of the customers. Thus, decision makers are often interested in solutions that achieve a good compromise between several objectives (Vilcot and Billaut, 2011).

According to Ruiz and Vázquez-Rodríguez (2010) and Almada-Lobo et al. (2008) production scheduling problems are multi-objective by nature, requiring that several conflicting objectives have to be considered at the same time. Therefore the main purpose of multi-objective approaches is to provide to decision makers a set of solutions corresponding to non-dominated objectives vectors, also called Pareto optimal solutions or non-dominated solutions (Vilcot and Billaut, 2011).

In order to extend the Graham's notation to encompass multi-objective scheduling problems T'kindt and Billaut (2006) proposed several additional notations. The notation $F_l\ (Z_1, Z_2, \dots, Z_k)$, is used when the objective is to minimise a linear combination of k objectives; $\#(Z_1, Z_2, \dots, Z_k)$, is used when a Pareto approach is sought; $Lex\ (Z_1, Z_2, \dots, Z_k)$, is when the decision maker is not authorized to make trade-offs between the objectives, i.e., when using lexicographical order and optimizing the

objectives one after the other; $\in (Z_1/Z_2, \ldots, Z_k)$ indicates that the goal is to minimise Z_1, subject to bounds on objectives $Z_2, \ldots, Z_k$.

2.1.4. Other Characteristics

A scheduling problem can be further classified according to other aspects, such as the nature of the problem data and the time at which data is known. If we classify scheduling problems in view of the nature of data it can be **deterministic** or **stochastic**. In deterministic problems all data are assumed to be fixed and well known in advance while in stochastic problems at least one parameter is a random variable, typically of known probability (T'kindt and Billaut, 2006). According to Framinan and Perez-Gonzalez (2015) stochastic problems are more realistic than their deterministic counterpart, allowing capturing part of the inherent variability present in many real-life manufacturing environments due to many unpredictable events (e.g. machine breakdowns, worker absenteeism and job changes). However, the number of studies focusing on deterministic problems is significantly larger than stochastic ones, mostly due to being easier to solve. An example of a stochastic problem in a job-shop environment can be seen in Zhang et al. (2012), in which the processing times of jobs are independent random variables with known distributions.

Other characteristics of scheduling problems have also been explored. For instance, Wang et al. (2015) investigated the **learning** and **deterioration effects**. The learning effect occurs when the production time of a given job is shorter after it is scheduled. This may occur when firms and employees perform a task over and over again, performing it increasingly more efficiently. The deterioration effect occurs when any delay in processing a job is penalized by incurring additional time for accomplishing the job. Both effects make start time of jobs dependent of processing times.

If the set of jobs available for scheduling does not change over time, the problem is called **static**. On the other hand, in a **dynamic** problem jobs arrive intermittently at the manufacturing system and the foregoing schedule has to be re-established in "real time" (T'kindt and Billaut, 2006). According to Baker and Trietsch (2009) although dynamic models would appear to be more important for practical applications, static models often capture the essence of dynamic systems. Thus static models have been studied more extensively, moreover as they have proved more tractable than dynamic models. A practical example of a dynamic system is present in remanufacturing[4] environments, where the number of returned products and the return time are factors that cannot be controlled. In this condition, new job(s) and non-started operations of existing jobs will have to be rescheduled (Gao et

[4] Remanufacturing is a form of product recovery process which requires the repair or replacement of worn out or obsolete components and modules (Gao et al., 2015).

al., 2015). Other approaches for scheduling problems in dynamic systems can be seen in Li et al. (2012) and Zhou et al. (2009).

Finally, it is worth noting that all problem characteristics mentioned in this subsection are not included in the three-fields standard classification of Graham et al. (1979), as it was devised for classifying deterministic scheduling problems.

2.2. Solution Methods for Scheduling Problems

According to Zobolas et al. (2008) due to the virtually unlimited number of different production environments, many variations of production scheduling problems can be found. However, in the literature studies have focused mainly on a limited number of classical (general) problems which, most of the times, cannot be directly applied to more complex manufacturing structures.

Although it is intended that modelling correctly reflects the real-life problem, adding all the problem characteristics may greatly increase the difficulty of solving it. Therefore, methods for solving scheduling problems have focused on more simplified problems, in order to obtain good solutions in reasonable computing times. Moreover, often decision-makers value more having a set of good solutions to choose from than having just the optimal solution. Thus, the development of flexible solution methodologies, which can be modified and applied to several different cases, is of critical importance for production management practice (Zobolas et al., 2008).

Scheduling problems typically belong to the class of combinatorial optimization problems (Blum and Roli, 2003). According to Papadimitriou and Steiglitz (1982) this class of problems can be defined as the set of all instances of a problem, with each instance being defined by a finite search space (namely, the set of all feasible solutions). The goal is trying to find the best possible solution(s) in the search space.

Most scheduling problems are NP-Hard (Framinan et al., 2014), therefore solving scheduling problems to proven optimality may only be practical for small/medium-sized instances. The difficulty of solving a scheduling problem typically depends on the number of jobs and corresponding operations, and resources. Some examples of NP-Hard problems are: parallel machines scheduling problem (Garey and Johnson, 1979), FSPs (Yagmahan and Yenisey, 2008), FFSPs (Xianpeng et al., 2009), JSPs (Garey et al., 1976) and FJSP with overlapping in operations (Fattahi et al., 2009). Among the few scheduling P-class problems, where polynomial time algorithms solving it to optimally have been found, there are the two-stage permutation FSP (Johnson, 1954) and the FJSP with two jobs (Brucker and Schlie, 1990) and makespan minimisation.

For solving scheduling problems, several approaches have been put forward which can be broadly separated into exact and heuristic. Exact approaches are able to solve problems to optimality and, due to the complexity of scheduling problems, are often more suitable for tacking smaller instances; heuristic approaches are used for solving problems approximately, typically employed when solving medium/large-sized instances. Thus, exact approaches are typically aimed at solving P-class scheduling problems, whereas heuristic approaches are often used to deal with NP-Hard scheduling problems. As follows, exact and heuristic methods for scheduling problems are reviewed, where heuristic approaches are further separated into constructive/improvement and metaheuristic.

2.2.1. Exact Methods

The main purpose of exact optimization techniques is obtaining and ensuring the optimal solution for a given problem is reached. According to Zobolas et al. (2008) the most common exact methods for scheduling problems are **branch algorithms** (namely, branch-and-bound – B&B), **Mixed Integer Programming (MIP)** and **decomposition methods**.

Cook (2012) describes the **B&B method** as the repeated application of a process for splitting a space of solutions into two or more subspaces and the adopting of a bounding mechanism to indicate if it is worthwhile to explore any or all of the newly created sub-problems. It consists of three main procedures: initialization, branching and bounding. Although the B&B algorithm does not generate all admissible solutions, they are implicitly evaluated.

There are other branching algorithms which combine B&B with other features such as cutting planes (Branch-and-Cut method – B&C) and column generation methods (Branch-and-Price method – B&P).

Concerning scheduling problems, B&B method is the preferred technique for solving FFSP optimally (Liao et al., 2012; Ruiz and Vázquez-Rodríguez, 2010). For instance Choi and Lee (2009) suggested a B&B algorithm to solve the two-stage hybrid flow-shop scheduling problem with the objective of minimising the total number of tardy jobs. The authors incorporated new methods to obtain lower bounds and three dominance properties that can reduce the search space. However, B&B method has also been applied to other variants of scheduling problems. Cheng et al. (2011) developed a B&B algorithm to solve the single-machine scheduling problem with deteriorating jobs and setup times to minimise the maximum tardiness. The algorithm was able to solve instances of up to 1000 jobs in reasonable time.

Some authors also have implicitly used branch techniques through mathematical programming, i.e. they represent their problem as an **MIP** model and use a regular solver for obtaining a solution (e.g.

Sun et al. (2010) for a case in steel manufacturing). MIP is also often used for obtaining lower bounds, thus enabling to evaluate the performance of heuristics (e.g. Saidi-Mehrabad and Fattahi, 2007; Seidgar et al., 2015).

Apart from the previously mentioned exact techniques there are two simple algorithms that allow solving to optimality special cases of scheduling problems in polynomial time.

One of them is the well-known **Johnson's Procedure** (Johnson, 1954) which minimises the makespan objective in the two-stage permutation FSP. The procedure starts by determining the minimum processing time on each stage. If the minimum processing time occurs on stage 1, the associated job is placed in the first available position in the sequence; otherwise, it is placed in the last available position in the sequence. This process is repeated until all the jobs are sequenced (Buffa and Sarin, 1987); afterwards, each job is scheduled in all stages at once according to the obtained job sequence.

The second one is the **Moore Procedure** (Moore, 1968) for solving the single machine scheduling problem with total number of tardy jobs minimisation. Firstly, jobs are arranged in increasing order of their due dates. If this sequence yields one or zero tardy jobs, then the schedule is optimal and the procedure stops. Otherwise, the first tardy job in the previous schedule is identified. Next, is selected the longest job from among the jobs whose due date is not superior to the job previously identified. The selected job is placed at the end of the sequence and job's completion times are revised (Buffa and Sarin, 1987). This procedure is iterated until there are no tardy jobs or cannot be improved further.

As mentioned previously, most scheduling problems have been proven to be NP-hard. So it can be too time consuming or even impractical to achieve the optimal solution with exact algorithms. As in most cases they need exponential computation times, they are impractical for large scale applications (Zobolas et al., 2008). Such situations require heuristic algorithms whose main objective is to obtain good quality solutions efficiently. These approaches can be further divided into constructive and improvement heuristics, and metaheuristic methods which have received increasing attention over the years.

2.2.2. Heuristic Methods: Constructive and Improvement

Contrary to exact methods, the computation requirements of heuristic approaches are relatively low and do not grow exponentially as the problem size increases (Huang and Süer, 2014). Therefore, heuristic methods are generally more effective for obtaining high quality solutions for large instances

in reasonable computing times; however, there is no guarantee of finding the global optimum value (Zobolas et al., 2008; Talbi, 2009).

Constructive heuristics involve building a new feasible solution, step by step according to a predetermined set of rules. These methods tend to choose the best option in each step and terminate when a complete solution is constructed (Martí and Reinelt, 2011; Murty, 2003). In scheduling problems the elements that are usually added in each step are operations of jobs (Zobolas et al., 2008). Although these heuristics can be the fastest in obtaining an admissible solution, the solutions quality usually leaves plenty of room for improvement (Zhang et al., 2007).

The most widely used constructive heuristics for scheduling problems are based on **dispatching (or priority) rules**. In general, they are used to select the next operation (out of a set of waiting operations) to be processed in a given available resource trough different priority parameters, e.g. processing times and due dates (Calleja and Pastor, 2014; Jayamohan and Rajendran, 2000). Some dispatching rules (e.g. FISFS, SPT, and EDD) may also be used for ordering the jobs in a sequence and then each job will be scheduled in all resources at once with a forward or backward scheduling approach. Forward scheduling approach means scheduling ahead from a point in time whereas backward scheduling means scheduling backward from a due date (Stevenson, 2005).

According to Sels et al. (2012) the attractiveness of dispatching rules is due to the fact that they are mostly intuitive in nature and can be easily implemented on the shop floor. Unfortunately, their solution quality is low due to the lack of flexibility. Table 2 summarizes most common and simple dispatching rules (SDR) for the ranking and assignment of jobs onto resources.

Table 2. Commonly used dispatching rules in scheduling problems (Askin, 1993 and Buffa, 1987).

Rule	Description
Shortest Processing Time (SPT)	Priority is given to the job with the shortest processing time on the resource under consideration.
Longest Processing Time (LPT)	Priority is given to the job with the longest processing time on the resource under consideration.
Earliest Due Date First (EDD)	Priority is given to the job with the earliest due date.
First In System First Served (FISFS)	Priority is given to the job that arrived in the shop floor first.
First Come First Served (FCFS)	Priority is given to processing of the job that arrived at the resource first.
Least Slack First (LSF)	Priority is given to the processing of the job that has least slack (the slack is the difference between the due date and the work remaining on the job).
Least Work Remaining (LWR)	Priority is given to the job with the least amount of total processing remaining to be done.
Most Operations Remaining (MOR)	Priority is given to the job with the most operations remaining in its processing sequence.
Most Work Remaining (MWR)	Priority is given to the job with the most total processing time remaining.
Random	A job is randomly chosen.

A variation of this approach is the composite dispatching rules (CDR), which have been receiving increased attention (Jayamohan and Rajendran, 2004). According to Tay and Ho (2008), CDR are heuristic combinations of SDR aiming inheriting their advantages. Empirical results show that with a fitting combination, CDR may perform better than SDR with regards to schedules quality. Ruiz and Vázquez-Rodríguez (2010) also add that dispatching rules are particularly suitable to deal with complex, dynamic and unpredictable environments, hence their popularity in practice.

The **NEH algorithm**, proposed by Nawaz et al. (1983), is a constructive heuristic commonly used in the scheduling literature. It was initially proposed to solve the permutation FSP with minimisation of makespan but over time it was adapted to solve other scheduling problems such as FFSP (e.g. Naderi et al., 2009; Ruiz et al., 2008). According to T'kindt and Billaut (2006) the NEH algorithm works as follow. Initially jobs are sorted by decreasing sums of processing times on resources. The heuristic considers only the first two jobs and retains the permutation schedule which has the minimal makespan value. This is the starting partial schedule. Then, it inserts the third job of the initial sorting, by trying all the possible positions in the partial schedule; the one which has the minimal makespan value is retained. This process is iterated until all the jobs are scheduled.

Another widely used heuristic with success in some variants of the scheduling problem, namely the JSP (Zobolas et al., 2008), is the **Shifting Bottleneck Procedure (SBP)**. This heuristic was proposed by Adams et al. (1988) for makespan minimisation and belongs to the class of machine-based decomposition methods (Pinedo, 2012). It works under the principle of giving full priority to the bottleneck stage (resource), maximising in this way its productivity, and consequently the productivity of the entire shop floor. Following Moursli and Pochet (2000), the SBP is described as follow. At each iteration, it schedules the bottleneck stage among the stages to be scheduled. The bottleneck stage (i.e. the one to be scheduled next) is the currently non-scheduled stage, which has the maximum makespan when scheduled. Every time a new stage is scheduled, some of the already scheduled stages might be rescheduled as their release dates might have changed – this rescheduling may lead to better solutions.

SBP has been adapted to handle: other process characteristics in job-shop environments such as sequence-dependent setup times (Balas et al., 2008); other manufacturing environments such as open-shop and assembly shops (Ramudhin and Marier, 1996); and other objective functions such as the total weighted tardiness (Braune and Zäpfel, 2016; Pinedo and Singer, 1999).

Improvement heuristics also known as local search (or neighbourhood) methods start from an initial feasible solution and iteratively try to improve it through successive small changes (named moves or neighbourhood operators) leading to obtaining new neighbouring solutions. However, two important aspects must be considered in local search methods (Al-Hinai and ElMekkawy, 2011). The first one is the size of neighbourhood and the second one is the feasibility of solutions. A solution is feasible only if it complies with all constraints of given problem. To deal with the feasibility issue there are two possible scenarios: only considers feasible moves; accepts all moves and then either rejects the infeasible moves or implement a repairing procedure.

According to Zobolas et al. (2008), most common moves in scheduling problems are the 2−Opt, the 1−1 exchange and the 1−0 exchange moves. The **2−Opt** move reverses a set of tasks of random length in a resource while the **1−1 Exchange** move swaps two tasks from the same resource. Finally, the **1−0 Exchange** move transfers a task from its position in one resource to another position in the same resource.

Lately, other moves also have been proposed and used in local search algorithms, mostly for FJSPs. Some of them consist in the following: changing the machine assignment (González et al., 2015; Lei and Guo, 2014); permuting the position of two adjacent operations (Jia and Hu, 2014; Yazdani et al., 2010); exchanging two randomly chosen operations (Lei and Guo, 2014; Li et al., 2014); and exchanging two consecutive operations on the same resource (Al-Hinai and ElMekkawy, 2011; Gao et al., 2007).

In improvement heuristics newly obtained solutions replace the previous one if they are of better quality. These methods end up in local optima (i.e. the best solution found in the neighbourhood of the initial solution) which rarely corresponds to the global optimum (Kuhpfahl & Bierwirth, 2016; Rajkumar et al., 2011).

2.2.3. Heuristic Methods: Metaheuristics

Metaheuristics are a class of heuristic methods which often integrates constructive and improvement methods. They are specifically designed to find satisfactory solutions of complex problems (NP-Hard), as is often the case in combinatorial optimization, with greater efficiency. Based on the metaheuristic definition by Stützle in 1999, Zobolas et al. (2008) state metaheuristics are an intelligent way to explore the solution space facilitating the escape from local optima. This escape often involves accepting worst solutions allowing a temporary degradation of the best solution obtained so far. Thereafter a local search in unexplored neighbourhoods may lead to even better solutions.

The search technique in metaheuristics often follows two global strategies. The first one is called exploration (diversification) and aims an effective exploration of all possible neighbourhoods of the solutions space (Zobolas et al., 2008). The second one is the exploitation (intensification) which is important for intensifying the search in some promising areas of the accumulated search experience (Boussaïd et al., 2013).

Metaheuristics can be classified according to different criteria. The most commonly used classification in the literature focuses on the number of current solutions carried from one iteration to the next, dividing metaheuristics into two classes: single-solution based metaheuristics and population-based metaheuristics (Boussaïd et al., 2013; Talbi, 2009). Examples of **single-solution based metaheuristics** are simulated annealing (SA), tabu search (TS), GRASP, variable neighbourhood search (VNS) and ILS. All of these have in common starting with a single initial solution and moving away from it, often describing a trajectory in the search space. On the other hand, **population-based metaheuristics** typically start from an initial population of solutions and iteratively apply the generation of a new population and the replacement of the current population (Talbi, 2009). Some examples are genetic algorithm (GA), ant colony optimization (ACO) and particle swarm optimization (PSO).

Although these two classes of metaheuristics operate differently in the search for the best solution, they are complementary. Single-solution based metaheuristics are exploitation oriented and population-based metaheuristics are exploration oriented (Talbi, 2009).

Concerning scheduling problems, over the years metaheuristics have been the preferred methods for obtaining good quality schedules in reasonable time for real industrial problems; moreover, as they typically also lead to better results than classical dispatching rules or greedy algorithms (Pezzella et al., 2008). Main metaheuristic implementations for solving scheduling problems in manufacturing environments will be briefly described as follows.

Simulated Annealing Algorithm

The SA algorithm was proposed by Kirkpatrick et al. in 1983 and is inspired by the annealing technique used by the metallurgists. It consists in carrying a material at high temperature, then in lowering this temperature slowly in order to obtain a ‘‘well ordered’’ solid state of minimal energy. The method transposes the annealing process to the solution of an optimization problem in the following way: the objective function of the problem, similar to the energy of a material, is minimised, by introducing a fictitious temperature which is a simple controllable parameter of the algorithm (Boussaïd et al., 2013).

According to Ramezani et al. (2015), the SA algorithm can be described as follow. It starts from an initial solution and continues producing and evaluating neighbourhood solutions in different stages, called temperatures. In each level of temperature a maximum number of movements are permitted to find a better solution; temperature which decreases until reaching a lower energy state. When a neighbourhood solution has a better objective function value compared with current solution, it is stored as the best found solution so far; otherwise, the neighbourhood solution is accepted according to an acceptance probability (see Figure 6). The initial temperature (T) should be sufficiently high to allow accepting more lower-quality solutions in the first iterations.

The fundamental idea of the SA algorithm is to allow moves resulting in solutions of worse quality than the current solution in order to escape from local minima (Blum and Roli, 2003).

```
Choose an initial solution s for the system to be optimized
Initialize the temperature T
while the stopping criterion is not satisfied
    repeat
        Randomly select s' ∈ N(s)
        if f(s') ≤ f(s) then
            s ← s'
        else
            s ← s' with a probability p(T, f(s'), f(s))
        end
    until the "thermodynamic equilibrium" of the system is reached
    Decrease T
end while
return the best solution found
```

Figure 6. Pseudocode of the SA algorithm (Boussaïd et al., 2013).

An application of SA can be seen in Dhouib et al. (2013) where the authors proposed a new simple algorithm for obtaining an initial solution and then they try to optimize it using SA. The problem addressed for the first time was the minimisation of the total number of tardy jobs in a permutation FSP with sequence-dependent setup times and time lags constraints – $Fm|prmu, ST_{sd}, \theta^{min}, \theta^{max}|\sum U_j$ according to Graham et al. (1979)' classification.

Naderi et al. (2009) suggested an improved SA for hybrid FSP with sequence-dependent setup and transportation times to minimise total completion time and total tardiness. According to the authors, previous SA implementations often demonstrate a premature convergence due to the lack of efficient mechanisms to diversify the search space. To overcome this weakness a new neighbourhood search structure was incorporated to achieve a compromise between intensification and diversification mechanisms, augmenting the competitive performance of their proposed SA.

Elmi et al. (2011) addressed the problem of scheduling parts in job shop cellular manufacturing systems. They considered exceptional parts' need to visit resources in different cells (resulting in intercellular moves) and re-entrant parts, which required visiting some resources more than once in a non-consecutive manner. This problem is named job-shop cell scheduling (JCS) with intercellular moves and re-entrant parts. The objective function is the minimisation of makespan. The authors developed a SA-based solution approach due to the complexity of the model. To increase the efficiency of the search algorithm, a neighbourhood structure based on the concept of blocks[5] was

[5] A block is a maximal sequence of adjacent critical operations that require to be processed on the same resource.

applied. The authors concluded that proposed SA is effective in reaching the optimal makespan for most instances.

Tabu Search Algorithm

The TS algorithm was introduced by Glover in 1986. According to Blum and Roli (2003) it explicitly uses the short-term history of the search, both to escape from local minima and to implement an explorative strategy. For this it uses a tabu list that keeps track of the most recently visited solutions (or attributes[6]) and forbids moves toward them. The neighbourhood of the current solution is thus restricted to the solutions that do not belong to the tabu list. At each iteration the best solution from the allowed set is chosen as the new current solution. Additionally, this solution is added to the tabu list and one of the solutions that were already in the tabu list is removed. The algorithm stops when a termination condition is met (see Figure 7).

1: Choose an initial solution s in the search space
2: $TabuList \leftarrow \emptyset$
3: **while** the stopping criterion is not satisfied
4: | Select the best solution $s' \in N(s) \setminus TabuList$
5: | $s \leftarrow s'$
6: | Update $TabuList$
7: **end while**
8: **return** the best solution found

Figure 7. Pseudocode of the TS algorithm (Boussaïd et al., 2013).

Saidi-Mehrabad and Fattahi (2007) state TS algorithms are more often used for solving JSPs and FJSPs. The authors presented a TS algorithm for solving the FJSP with sequence-dependent setup times minimising makespan. It is based on a hierarchical approach: a procedure searches for the best sequence of job operations, and another procedure finds the best choice of resource. Results indicate that this algorithm can produce optimal solutions in a short computational time for small and medium sized problems. Furthermore, the authors also state that it can be easily applied in real-world factory conditions and for large size problems.

According to Zhang et al. (2007) neighbourhood structures and move evaluation strategies play the central role in the effectiveness and efficiency of TS for JSPs. The authors therefore construct a new enhanced neighbourhood structure which can avoid cycling and investigate much larger solution

[6] Attributes are usually components of solutions, moves, or differences between two solutions.

spaces. It was applied for solving the JSP with minimisation of makespan. The effectiveness of the proposed neighbourhood structure was validated through testing on a set of benchmark instances, in which a large number of upper bounds were improved.

Variable Neighbourhood Search Algorithm

VNS is a simple and effective metaheuristic with a different mechanism than other single-solution based metaheuristics (e.g. SA and TS algorithms) for diversifying the search. It was created by Mladenović and Hansen (1997) and involves a systematic change of neighbourhood within a local search algorithm, thereby avoiding entrapment at a local optimum. According to the authors VNS metaheuristic explores increasingly distant neighbourhoods of the current incumbent[7] solution, and jumps from there to a new one only if an improvement is achieved. This way, favourable characteristics of the incumbent solution (e.g. some variables already at their optimal value) are often kept and used to obtain promising neighbouring solutions. Moreover, a local search routine is applied repeatedly to get from these neighbouring solutions to local optima (see complete procedure in Figure 8).

```
1:  Select a set of neighbourhood structures N_n, n = 1, ..., n_max
2:  Choose, at random, an initial solution s in the search space
3:  while the stopping criterion is not satisfied
4:  |  n ← 1
5:  |  while n < n_max
6:  |  |  Shaking: select a random solution s' in the n^th neighbourhood N_n(s) of s
7:  |  |  Apply a local search starting from s' to get a solution s''
8:  |  |  if s'' is better than s then
9:  |  |  |  s ← s''
10: |  |  |  n ← 1
11: |  |  else
12: |  |  |  n ← n + 1
13: |  |  end if
14: |  end while
15: end while
16: return the best solution found
```

Figure 8. Pseudocode of the VNS algorithm (Boussaïd et al., 2013).

[7] The best feasible solution found so far.

Blum and Roli (2003) concluded that the process of changing neighbourhoods in case of no improvements corresponds to a diversification of the search, and the choice of neighbourhoods of increasing cardinality yields a progressive diversification.

VNS has been used to solve a considerable diversity of scheduling problems. According to the three-field notation of Graham et al. (1979), some examples are $Fm|batch(b)|\sum(w_j)T_j$, T_{max}, $\sum(w_j)U_j$ (Lei and Guo, 2011), $Fm|prmu|\sum(w_j)F_j$ (Costa et al., 2012), $Pm|r_j, ST_{sd}, prec|\sum w_j T_j$ (Driessel and Mönch, 2011), $1|ST_{sd}|\sum w_j T_j$ (Kirlik and Oguz, 2012), $FJm|prec, ST_{sd}|F_l(C_{max}, \overline{T})$ (Bagheri and Zandieh, 2011) and $Jm|prec, ST_{sd}|C_{max}$ (Roshanaei et al., 2009).

Genetic Algorithm

GA is an optimization algorithm developed by John Holland in 1975 which intends imitating the evolution of living things based on the process of natural selection (Ishikawa et al., 2015).

Following Pezzella et al. (2008), GA is described as follow. Starting from an initial population of individuals (chromosomes), genetic operators are applied (e.g. selection, reproduction, rearrangement, and mutation) in order to produce offspring (a new population of individuals), which are presumably more fit than their ancestors. At each generation (iteration), every new individual corresponds to a solution and the individuals whose fitness is higher are most likely to survive selection as well as being chosen for reproduction (see Figure 9). The selection of individuals for reproduction has a special role since it is one of the factors that determines the evolutionary search spaces (Türkyılmaz and Bulkan, 2015).

```
1: Initialize the population with random individuals
2: Evaluate each individual
3: repeat
4:     Select parents
5:     Recombine pairs of parents
6:     Mutate the resulting offspring
7:     Evaluate new individuals
8:     Select individuals for the next generation
9: until a termination condition is satisfied
```

Figure 9. Pseudocode of the GA algorithm (Boussaïd et al., 2013).

There are some aspects of GA that must be taken into consideration in order to improve their efficiency. One of them is the selection of a string format for the individuals, which is a very important step for a successful implementation (Chen et al., 1999). If the chromosomal representation is well designed, no infeasible schedules will be produced after recombination, and the algorithm can potentially be more efficient. Another important aspect is the initial population of the GA, as it can affect the convergence speed and the quality of the final solution(s) (Rahnamayan et al., 2007; Türkyılmaz and Bulkan, 2015).

GA has proven to be a powerful technique for combinatorial optimization problems and has been successfully used to solve scheduling problems, namely FJSPs (Ishikawa et al., 2015). Demir and İşleyen (2014) proposed a GA to solve the FJSP with overlapping in operations and minimisation of makespan. For solutions' representation it was used the chromosome representation by Zhang et al. (2011), consisting of a string with two components: machine selection and operation sequence. Additionally, the authors developed a new search methodology which was applied in the generation of the initial population, specifically in the operations sequence part. At the same time, an efficient decoding[8] methodology was also adopted in order to reduce the search space. The computational study showed that their algorithm surpassed other known algorithms for the same problem.

Swarm Intelligence Algorithms

According to Talbi (2009) swarm intelligence algorithms were inspired from the collective behaviour of species such as ants, bees, wasps, termite, fish, and birds. Among the most successful swarm intelligence inspired optimization algorithms are **ACO** and **PSO**. These algorithms are distinctly different from other population-based metaheuristics (e.g. GA) as they do not use the filtering operation (such as crossover and/or mutation). However, the members of the entire population are maintained through the search procedure in order to socially share the information among individuals and, thus direct the search towards the best position in the search space (Tasgetiren et al., 2007). Among swarm intelligence algorithms, PSO appears to be most common in the scheduling literature (e.g. Sha and Hsu, 2008; Tadayon and Salmasi, 2012) followed by ACO (e.g. Neto and Filho, 2011; Yagmahan and Yenisey, 2008).

[8] Decoding is to convert the coded solution to a feasible scheduling solution (e.g. a scheduling Gantt chart) (Gao et al., 2015).

Other Metaheuristics

Other metaheuristics have also been applied in scheduling problems, although less frequently: GRASP (Molina-sánchez and González-Neira, 2016; Rajkumar et al., 2011); ILS (e.g. Mousakhani, 2013; Subramanian et al., 2014); scatter search/path relinking (e.g. González et al., 2015; Rahimi-Vahed et al., 2008); artificial bee colony (e.g. Gao et al., 2015; Wang, 2012); and harmony search (e.g. Gao, et al., 2015; Zammori et al., 2014).

Blum and Roli (2003) state that a balance between diversification and intensification during the search procedure is important; on one side to quickly identify regions in the search space with high quality solutions and, on the other side, to avoid wasting too much time in regions of the search space which are either already explored or which do not provide high quality solutions. Therefore a current trend is the hybridization of heuristic methods to merge the strengths and eliminate the weaknesses of different metaheuristic concepts and consequently finding solutions of higher quality. According to Zobolas et al. (2008) most hybrid heuristic methods for scheduling problem consists in the hybridization of population-based metaheuristics with local search methods, where various components and solution characteristics are shared among two or more heuristic approaches. Some examples of hybrid heuristics methods more frequently used in scheduling problems can be found in Table 3.

Table 3. Some examples of hybrid heuristics methods used in scheduling problems.

Main method	Secondary method	Publication	Problem addressed
GA	SA	Shahsavari-Pour and Ghasemishabankareh (2013)	$FJm\|prec, recrc\|$ $\#(C_{max}, \sum W_k, W_{max})$
		Wang and Zheng (2001)	$Jm\|prec\|C_{max}$
	VNS	Gao et al. (2008)	$FJm\|prec, recrc\|$ $Lex(C_{max}, \sum W_k, W_{max})$
		Türkyılmaz and Bulkan (2015)	$FJm\|prec, recrc\|\sum T_j$
	Local search	Al-Hinai and ElMekkawy (2011)	$FJm\|prec\|C_{max}$
		Ruiz et al. (2006)	$Fm\|prmu\|C_{max}$
PSO	VNS	Gao et al. (2015)	$Jm\|prec\|C_{max}$
		Tasgetiren et al. (2007)	$Fm\|prmu\|C_{max}, \sum F_j$
	TS	Zhang et al. (2009)	$FJm\|prec, recrc\|$ $F_l(C_{max}, W_{max}, \sum W_k)$
	SA	Shao et al. (2013)	$FJm\|prec, recrc\|$ $\#(C_{max}, W_{max}, \sum W_k)$
		Xia and Wu (2005)	$FJm\|prec, recrc\|$ $F_l(C_{max}, \sum W_k, W_{max})$
TS and VNS		Li et al. (2010)	$FJm\|prec\|F_l(C_{max}, \sum W_k, W_{max})$
TS and SA		Zhang et al. (2008)	$Jm\|prec\|C_{max}$

In Table 3, the first column corresponds to the main method and the second column corresponds to the secondary method. The fourth and fifth columns indicate the source publication and problem addressed herein. Hybrid heuristic methods appear to have been more frequently used in FJSP. This is due to being a problem with high applicability although very difficult to solve, for which these methods have shown higher quality results in low computing times. Among the heuristics proposed for $Jm|prec|C_{max}$ in Table 3, PSO-VNS from Gao et al. (2015) and GA-SA from Wang and Zheng (2001) have shown a better performance concerning ratio of the deviation (Gap) with respect to the best known solution. Both were able to find optimal solutions for almost all test instances.

Concerning multi-objective FJSP with minimization of C_{max}, W_{max} and $\sum W_k$ using Pareto approach, PSO-SA method by Shao et al. (2013) get to obtain more number of non-dominated solutions of high quality than other methods. Apart from Pareto approach previous method together with TS-VNS one by Li et al. (2010) have shown its superiority to other approaches.

Chapter 3

Problem Description

Currently *Softinov APS* is implemented in several types of industries; e.g. ceramic, automotive components, special steel alloys, moulds and conveyors manufacturers, pharmaceutical companies, and packaging industry. Despite each one of these production processes having its own scheduling specifications most of them have in common several aspects. Based on the most common scheduling requirements, *Softinov APS* requires handling a deterministic, static and multi-objective FJSP with several processing characteristics, such as release dates, precedence constraints, resources capacity (and availability), setup times, sub-resources, maximum and minimum waiting times between operations (time lags), among others. Out of all these possible characteristics, this work will focus on some of the most common and important ones; the corresponding problem is described as follows.

There is a set J of n jobs that have to be scheduled to a set R of m resources. Each resource $R_k \in R$ with $k \in \{1, \dots, m\}$ is identified by a code and has a priority (lower values correspond to higher priorities) for a given operation (from a job). Resources may be of two types: homogeneous work centres or normal work centres. Thus, set R consists of two resource subsets which are subset R^H (homogeneous work centres) and subset R^N (normal work centres). A homogenous work centre (HWC) could contain one or more identical resources having similar characteristics, such as performing the same kind of operations and having the same execution and setup times. Furthermore, a HWC has capacity distribution, i.e. it divides its capacity (number of available resources within HWC) for production of different jobs (see Figure 10a). On the other hand, a normal work centre (NWC) has no capacity distribution and can be seen as a single resource (see Figure 10b and 10c).

Still concerning resource capacity distribution, a HWC always has finite capacity but a NWC may have finite or infinite capacity; with finite capacity only one operation can be executed at a given time (see Figure 10b) whereas infinite capacity NWC can perform more than one operation at the same time (see Figure 10c).

3. Problem Description

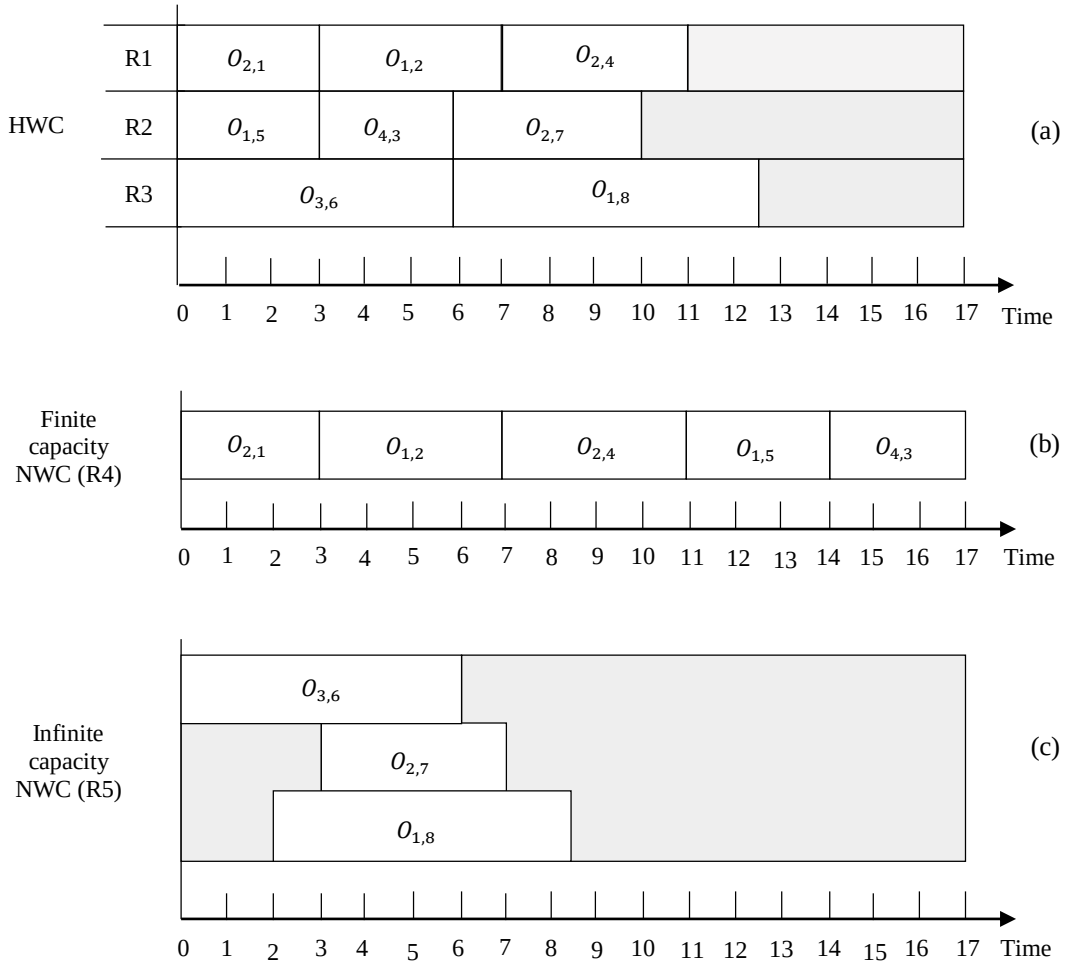

Figure 10. Examples of distribution capacity in (a) HWC with three resources, i.e. capacity = 3, (b) finite capacity NWC, and (c) infinite capacity NWC.

Each resource $M_k \in M$ has its own calendar for a given scheduling horizon, which includes working periods, non-working periods (e.g. weekends and holidays), and stopping periods (e.g. preventive maintenance). Capacity change may also occur during working periods, i.e. execution capacity relative to standard execution capacity may increase or decrease.

Regarding HWC, capacity changes in working periods result in an increase or decrease of available resources within it. In case of NWC (excluding batch resources), an increase/decrease of capacity is typically associated with a corresponding increase/decrease of processing speed. This last kind of capacity change is characterized by coefficient (1).

$$\text{Processing coefficient} = \frac{\text{Standard processing capacity}}{\text{Real processing capacity}} \quad (1)$$

Each job $j \in J$ with $j = \{1, \dots, n\}$ is characterized by a code, a priority (smaller value means higher priority), a due date d_j, a release date r_j (the smaller release date of all its operations) and a planned production quantity. Jobs can be independent or dependent of each other, as some correspond to finished goods and others to components for other jobs. Thus, precedence constraints between jobs may exist. Furthermore, each job j is composed of a predetermined sequence of H_j operations, also leading to precedence constraints between operations of the same job. However, it is possible to execute several operations of the same job at the same time if such operations are independent. The operation h of job j is denoted O_{hj} with $h \in \{1, \dots, H_j\}$ and can be processed by any resource of set $R_{hj} \subseteq R$ which contains the resources capable of performing O_{hj}.

Each operation O_{hj} can only be processed after its release date r_{hj} and the biggest completion time of its preceding operations have been reached. Once started, an operation can be completed with interruption due to pre-defined stopping periods and non-working periods in resources calendars. The completion time depends on its starting time, setup time, processing time and resource downtimes (when existing). The processing time of an operation O_{hj} when executed on resource $R_k \in R$ with $k \in \{1, \dots, m\}$ is represented as P_{hjk}. If the assigned resource is a HWC, i.e. $R_k \in R^H$, the operation processing time is equal to the standard processing time. In case it is executed on a NWC resource, its processing time may be different from standard one. This situation may occur because the latter does not consider capacity changes – it usually represents the normal processing time of an operation for a normal resource capacity. Therefore, if processing time of a given operation coincides with a capacity change period, it has to be updated according to equation (2).

$$\text{Real processing time} = \text{Processing coefficient} \times \text{Standard processing time} \quad (2)$$

Regarding setup times, each resource has associated a setup characteristic (e.g. colour, size) which could have its own setup matrix (except for infinite capacity NWC). It contains the variation of setup times for the transition between each pair of different possible combinations of a setup characteristic. If an operation is processed in an infinite capacity NWC or it is the first operation to be executed in a given resource, only the standard setup time is considered (i.e. the setup time is sequence-independent). For the other types of resources when an operation is performed after another operation, its setup time is sequence-dependent, i.e. it could depend on resource's setup matrix and the standard setup time. Therefore, both sequence-dependent and sequence-independent setup times can be considered.

3. Problem Description

Overall, the problem consists in assigning each operation O_{hj} to an eligible resource from set R_{hj} and determining its starting and completion times in order to minimise the total number of tardy jobs. A job j is tardy if the completion time of its last operation $O_{H_j j}$ is bigger than its due date d_j.

For modelling the problem, adding to previously defined notation, the following is used:

- j, v, t index the jobs, with $j, t \in \{1, \dots, n\}$ and $v \in \{0, \dots, n\}$ where 0 is a dummy job;
- g indexes the operations of job v where $g \in \{1, \dots, H_v\}$;
- f indexes the operations of job t where $f \in \{1, \dots, H_t\}$;
- S_{hjgvk} is the setup time for O_{hj} after processing O_{gv} in resource $R_k \in R$;
- e_{hjk} is 1 if resource R_k is eligible to process operation O_{hj}, 0 otherwise;
- a_{fthj} is 1 if O_{ft} is a precedence operation of O_{hj}, 0 otherwise;
- b_k is 1 if resource R_k has finite capacity, 0 otherwise;
- L is a large positive number.

The following decision variables are defined:

- X_{hjgvk} is a binary variable equal to 1 if O_{hj} is processed after O_{gv} in resource $R_k \in R$, where $O_{hj} \neq O_{gv}$;
- U_j is a binary variable equal to 1 if $O_{H_j j}$ is completed after due date of job j;
- C_{hj} is a continuous variable for the completion time of O_{hj}.

Extending the MILP model from Mousakhani (2013), the problem is formulated as follows.

$$\text{Min } Z = \sum_{j=1}^{n} U_j \tag{3}$$

Subject to:

$$\sum_{g=1}^{H_v} \sum_{v=0}^{n} \sum_{k=1}^{m} X_{hjgvk} = 1 \qquad \forall j \in \{1, \dots, n\},\ \forall h \in \{1, \dots, H_j\}, \tag{4}$$

$$\sum_{g=1}^{H_v} \sum_{v=0}^{n} X_{hjgvk} \leq e_{hjk} \qquad \forall j \in \{1, \dots, n\}, \forall h \in \{1, \dots, H_j\}, \forall k \in \{1, \dots m\}, \tag{5}$$

$$\sum_{h=1}^{H_j} \sum_{j=1}^{n} \sum_{k=1}^{m} X_{hjgvk} \leq 1 \qquad \forall v \in \{0, \dots, n\},\ \forall g \in \{1, \dots, H_v\}, \tag{6}$$

$$\sum_{h=1}^{H_j}\sum_{j=1}^{n} X_{hj10k} \leq 1 \qquad \forall k \in \{1, \dots m\}, \tag{7}$$

$$\sum_{h=1}^{H_j}\sum_{j=1}^{n} X_{hjgvk} \leq \sum_{h=1}^{H_j}\sum_{j=0}^{n} X_{gvhjk} \qquad \forall v \in \{0, \dots, n\}, \forall g \in \{1, \dots, H_v\}, \forall k \in \{1, \dots m\}, \tag{8}$$

$$C_{hj} \geq max_{t\in\{1,\dots,n\}, f\in\{1,\dots,H_t\}}\{C_{ft} a_{fthj}\} + \sum_{g=1}^{H_v}\sum_{v=0}^{n}\sum_{k=1}^{m} X_{hjgvk}\left(S_{hjgvk} + P_{hjk}\right) \tag{9}$$

$$\forall j \in \{1, \dots, n\}, \forall h \in \{1, \dots, H_j\},$$

$$C_{hj} \geq \left[C_{gv} + \sum_{k=1}^{m} X_{hjgvk}\left(S_{hjgvk} + P_{hjk}\right) - L\left(1 - \sum_{k=1}^{m} X_{hjgvk}\right)\right] \times \sum_{k=1}^{m} X_{hjgvk} b_k \tag{10}$$

$$\forall j \in \{1, \dots, n\}, \forall v \in \{0, \dots, n\}, \forall h \in \{1, \dots, H_j\}, \forall g \in \{1, \dots, H_v\},$$

$$C_{hj} \geq r_{hj} + \sum_{g=1}^{H_v}\sum_{v=0}^{n}\sum_{k=1}^{m} X_{hjgvk}\left(S_{hjgvk} + P_{hjk}\right) \tag{11}$$

$$\forall j \in \{1, \dots, n\}, \forall h \in \{1, \dots, H_j\},$$

$$C_{H_j j} - \sum_{g=1}^{H_v}\sum_{v=0}^{n}\sum_{k=1}^{m} d_j X_{H_j jgvk} \leq LU_j \qquad \forall j \in \{1, \dots, n\}, H_j \geq 1, \tag{12}$$

$$C_{hj} > 0 \qquad \forall j \in \{1, \dots, n\}, \forall h \in \{1, \dots, H_j\}, \tag{13}$$

$$X_{hjgvk} \in \{0,1\} \qquad \forall j \in \{1, \dots, n\}, \forall h \in \{1, \dots, H_j\}, \forall v \in \{0, \dots, n\}, \forall g \in \{1, \dots, H_v\}, \forall k \in \{1, \dots, m\}, \tag{14}$$

$$U_j \in \{0,1\} \qquad \forall j \in \{1, \dots, n\}. \tag{15}$$

Where $C_{0j} = C_{10} = 0$.

Objective function (3) aims minimising the total number of tardy jobs. Constraints (4) ensure that every operation is scheduled exactly once. Thus, this constraint set fulfils two decision levels that are being considered simultaneously: assignment and sequencing. Moreover, it makes that each operation has exactly one preceding operation (in its assigned resource).

Constraints (5) specify that each operation is assigned to one of its eligible resources. Constraints (6) ensure that every operation can have at most one succeeding operation (in its assigned resource),

accounting for the operation in the last position of each resource having no succeeding operations. In case of infinite capacity NWC resources, constraints (6) do not hold.

Constraints (7) guarantee the dummy operation is the first operation on resources and constraints (8) ensure that only operations on the same resource can be consecutive operations.

With constraints (9) an operation cannot be processed without is precedence operations being completed. Therefore, the completion time of a given operation should be bigger than the maximum completion time of its precedence operations plus setup and processing times.

Constraints (10) ensure that a finite capacity resource cannot process two different operations simultaneously. In other words, the difference between the completion times of two consecutive operations (on a finite capacity resource) should be greater than the setup time plus the processing time of the operation processed later. In case of infinite capacity resources these constraints do not hold.

Due to constraints (11) a succeeding operation of a job can only start from its release date onwards. Together, constraints (9), (10) and (11) make that the earliest time a job can start its succeeding operation(s) is when both the operation and the resource are available.

Finally, constraints (12) allow identifying the tardy jobs and constraints (13)-(15) define the decision variables.

Chapter 4

Heuristic Approaches

The general FJSP is an extension of the classic JSP which has proven to be NP-hard (Garey et al., 1976). Therefore, the FJSP with all the characteristics addressed in the present work is also an NP-hard problem. This has lead to the development of heuristic approaches, moreover as they may be more fitting to be used in the *Softinov APS* due to typically lower running times.

In this section three constructive methods and five improvement heuristics are proposed, and two metaheuristics (GRASP and ILS) are adapted to tackle the addressed problem. Constructive methods are applied to obtain initial solutions, which are then improved with the improvement heuristics. Concerning the metaheuristic approaches, these combine the characteristics of constructive and improvement methods to handle this hard combinatorial optimization problem in a more useful and efficient way.

The problems belonging to the class of FJSPs can be seen as the combination of two sub-problems: assignment and sequencing. In this way there are two main classes of approaches to solving them: hierarchical and integrated approaches. In hierarchical approaches the two sub-problems are solved independently (i.e. are separated) whereas in integrated ones the assignment and sequencing are solved simultaneously. Most of solution methods in FJSPs literature are based on hierarchical approaches. The main reason for this may be that obtaining solutions with integrated approaches is often more difficult; however, they usually allow achieving better results (Amiri et al., 2010; Pezzella et al., 2008). In this work, all heuristics are based on integrated approaches. Concerning scheduling rules, the forward scheduling rule is employed instead of the backward rule, in order to prevent operations from being scheduled before the start of the scheduling horizon.

4.1. Constructive Heuristics

The heuristics detailed in this section are used to reach an initial solution for the previously described problem. Three algorithms are proposed: job-by-job (JBJ), operation-by-operation (OBO) and resource-by-resource (RBR).

The working principle is similar across all the constructive heuristics. In each iteration, a job's operation is selected to be scheduled from a set of eligible operations based on simple dispatching rules. This is to be performed repeatedly until a complete solution (schedule) is constructed. The most noticeable difference between the three algorithms has to do with the way the resource for processing each operation is chosen.

As it is intended finding a schedule which minimises the total number of tardy jobs, the most commonly used dispatching rule is EDD. However, if two or more operations share the same due date, other dispatching rules are employed.

The three proposed constructive algorithms are detailed as follows.

4.1.1. Job-by-job Constructive Heuristic

This constructive algorithm starts by ordering all jobs based on due dates, from earliest to latest. Then, to ensure all precedence constraints between jobs are obeyed, jobs are reordered according to its precedence jobs (when existing).

In the next step, one job is selected from the top of the sorted list and all of its operations have to be scheduled, then proceeding to the following job in the list. Afterwards, it is necessary to check which operations from the selected job can be scheduled. This set of eligible operations E is composed of operations without precedence relationships or for which precedence operations have been completed. The operation with the earliest release date (ERD) is taken from E. If more than one operation has in common the same ERD, the operation that comes first is chosen. Afterwards, in a cyclic manner, it is assigned a resource from the set of eligible resources (when the operation can be performed in more than one eligible resource).

Thereafter, the operation starting and completion times are determined. If the assigned resource is a finite capacity NWC or a HWC, the operation starting time has to be bigger or equal than: (a) the earliest idle time of the resource; and (b) the release date and maximum completion time of its precedence operations. In case of a HWC, it is further necessary to assign a resource within HWC – the one with the earliest idle time interval is selected to perform the operation. In case of a tie, the

first resource found with the earliest idle time interval is used. If the resource is an infinite capacity NWC, the operation starting time only needs to be equal to or bigger than its release date and maximum completion time of precedence operations. The completion time is obtained take into account the operation starting time, standard setup time and operation processing time (which depends on resource capacity, including downtimes).

Operations once scheduled, are removed from E and added to set of scheduled operations S. This is to be performed until E is empty. In this case, the algorithm looks for operations from the selected job which can be scheduled, and proceeds until all operations have been scheduled. An overview of the JBJ constructive heuristic's pseudocode can be seen in Figure 11.

```
1:  Input: All required data about jobs to be scheduled and resources
2:  E ← ∅; S ← ∅
3:  Order jobs by due date (from the earliest to latest)
4:  Reorder jobs according to its precedence jobs
5:  for each job j in J
6:      while j has operations to be scheduled
7:          Copy eligible operations to E
8:          while E ≠ ∅
9:              Select one operation O_hj from E according to ERD rule
10:             Cyclically assign a resource from set R_hj
11:             Determine the starting time of O_hj
12:             Determine the completion time of O_hj taking into account standard setup time
                and resources capacity (including downtimes)
13:             Add O_hj to S and remove from E
14:         end while
15:     end while
16: next
17: UpdateSetupTimes(S)
18: Output: The complete schedule
```

Figure 11. Pseudocode of the JBJ constructive heuristic.

When all operations have been scheduled, their starting and completion times must be updated. This is to obtain the correct setup times, which is only possible once the complete scheduling of operations is known. The previously used standard setup times can now be maintained, reduced or increased. This procedure, UpdateSetupTimes(), is also performed in the other constructive algorithms in the following subsections.

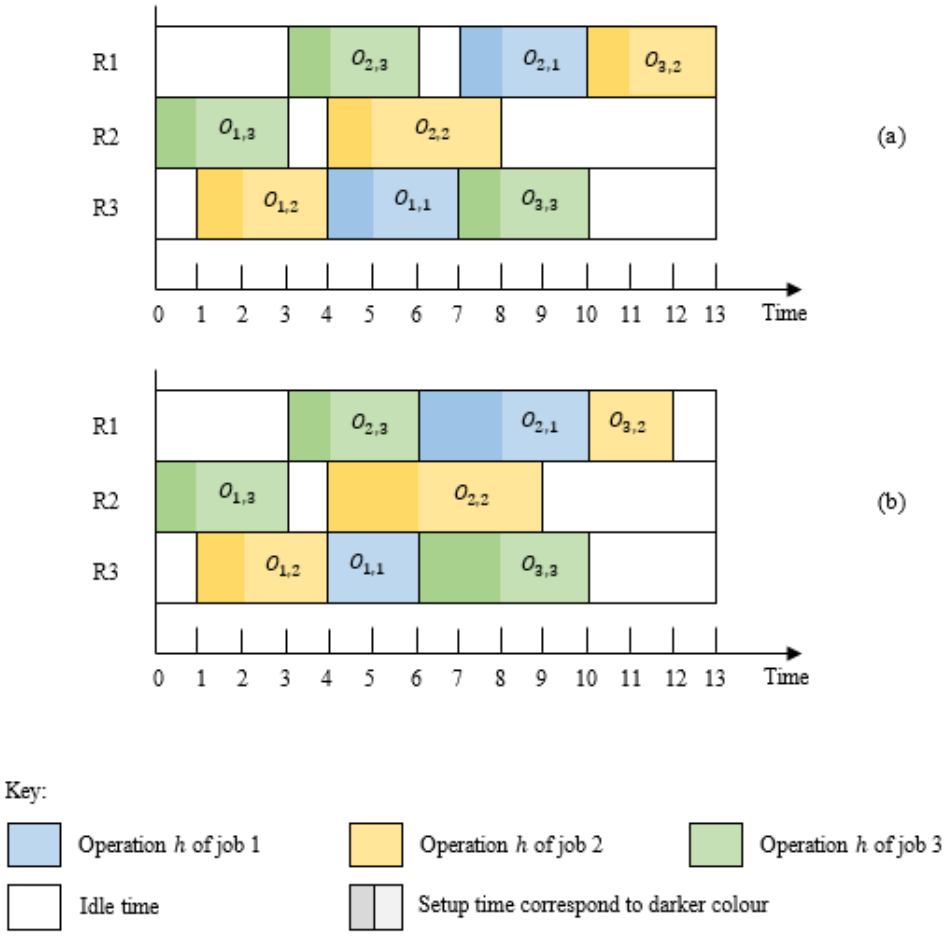

Figure 12. An example of a schedule (a) before and (b) after applying UpdateSetupTimes().

An application example of the UpdateSetupTimes procedure can be seen in Figure 12, showing an instance composed of 3 jobs and 3 resources (3x3) before and after the procedure. For example, the completion time of operation $O_{1,1}$ was updated from 7 to 6. This occurred due to operations $O_{1,2}$ and $O_{1,1}$ having the same setup characteristic thus not requiring any setup time prior to $O_{1,1}$. The main data of the 3x3 instance used in above example are available in Appendix A.

4.1.2. Operation-by-operation Constructive Heuristic

The following constructive method was inspired in the heuristic approaches used in the recent works of Calleja and Pastor (2014) and Zhang and Yang (2016). An advantage of this method compared with JBJ is that in each iteration any eligible operation from any job is a candidate to be selected for scheduling. However, it may be more time consuming in the step where a resource is chosen to perform a selected operation.

The OBO constructive heuristic can be described as follows. At each iteration, the algorithm selects the eligible operations (set E) from all operations that still remain to be scheduled (set U). Thereafter, one eligible operation is selected to be scheduled according to the EDD rule; in case of ties the eligible operation with the ERD is selected.

From the set of resources which can perform the selected operation, the one which allows the earliest operation starting time is chosen. This way, resources idle times and operations waiting time are reduced, simultaneously attempting avoiding operation's tardiness. When the earliest starting time is the same across the eligible resources, the resource in which the operation is completed earlier is chosen.

Finally, the operation is placed into the set of scheduled operations S, removed from E and the algorithm moves eligible operations from U to E again. This is performed until all operations have been scheduled. An overview of the OBO constructive heuristic's pseudocode can be seen in Figure 13.

1: **Input:** All required data about jobs to be scheduled and resources
2: $U \leftarrow$ all job's operations to be scheduled
3: $E \leftarrow \emptyset$; $S \leftarrow \emptyset$
4: **while** $U \neq \emptyset$
5: Move eligible operations from U to E
6: Select an operation O_{hj} from E according to EDD rule.
In case of ties, choose the operation according to ERD rule.
7: Determine the starting time of O_{hj} in each resource from R_{hj}
8: Determine the completion time of O_{hj} taking into account standard setup time and resources capacity (including downtimes).
9: Assign the resource with the earliest operation starting time.
In case of ties, choose the resource in which O_{hj} will be completed earlier.
10: Add O_{hj} to S and remove from E
11: **end while**
12: UpdateSetupTimes(S)
13: **Output:** The complete schedule

Figure 13. Pseudocode of the OBO constructive heuristic.

4.1.3. Resource-by-resource Constructive Heuristic

This constructive heuristic was inspired by the algorithm used in Türkyılmaz and Bulkan (2015) for initial population generation. As with the OBO method, initially, all operations (of all jobs) are candidates to be selected for scheduling.

In the original version of the algorithm, a resource available at a specific instant t is chosen randomly, and a non-scheduled operation that can be processed by this resource is selected to be scheduled according to a dispatching rule. At the end of the iteration, instant t is incremented one unit and the algorithm proceeds until all operations are scheduled. A disadvantage of the method is that incrementing t one by one is often not efficient. To overcome this limitation, an adaptation of the method is proposed; an overview of the new heuristic, named RBR, is given in Figure 14.

Iteratively, the algorithm goes through each resource and selects from the set of operations to be scheduled U the eligible operations which can be performed by it. If there is no eligible operation to be performed, the algorithm proceeds to the next resource. Otherwise, it selects an eligible operation according to a set of dispatching rules, applied in the following order:

Rule 1: Select the eligible operation with the earliest due date;

Rule 2: Select the eligible operation with the earliest release date;

Rule 3: Select the eligible operation with the shortest standard processing time;

Rule 4: Select the eligible operation that comes first.

Once the operation has been chosen, its starting and completion times are determined based on the resource earliest idle time interval to which the operation may be allocated. After going through each resource exactly once, the resources are ordered by increasing order of their total workload. The goal is to balance workload over them and consequently, avoid their overload. The procedure ends when all operations have been scheduled.

```
1:  Input: All required data about jobs to be scheduled and resources
2:  U ← all job's operations to be scheduled
3:  E ← ∅; S ← ∅
4:  while U ≠ ∅
5:      Order all resources by increasing order of total workload
6:      for each resource R_k in R
7:          if U = ∅ then exit for end if
8:          Copy eligible operations that can be processed by resource R_k from U to E
9:          if E ≠ ∅ then
10:             Select an operation O_hj from E by applying orderly the set of dispatching rules
                (Rule 1, Rule 2, Rule 3 and Rule 4)
11:             Assign R_k to O_hj
12:             Determine the starting time of O_hj
13:             Determine the completion time of O_hj taking into account standard setup time
                and resources capacity (including downtimes)
14:             Add O_hj to S and remove from U
15:             E ← ∅
16:         end if
17:     next
18: end while
19: UpdateSetupTimes(S)
20: Output: The complete schedule
```

Figure 14. Pseudocode of the RBR constructive heuristic.

4.2. Improvement Heuristics

In this section five improvement heuristics based on simple move operators (reassign, external exchange, internal exchange, swap, and reinsert-reassign) are presented in order to search for better neighbourhood solutions.

The reassign and external exchange improvement heuristics are performed to modify (i) assigned resources to operations, whereas internal exchange and swap ones change (ii) the sequence order of scheduled operations. Additionally, reinsert-reassign looks to improve (i) and (ii) simultaneously.

Figure 15 shows the pseudocode used for all improvement heuristics. The main difference among them lies in the move operator used. Note that for all improvement methods a move can only be applied if the resulting solution is feasible.

```
1:  Input: An initial solution sol_i
2:  bestSolution ← sol_i
3:  improvements ← true
4:  while improvements
5:      improvements ← false
6:      for each feasible move
7:          solution ← bestSolution
8:          ApplyMoveOperator(solution)
9:          Reschedule(solution)
10:         if f(solution) < f(bestSolution) then
11:             bestSolution ← solution
12:             improvements ← true
13:             exit for
14:         end if
15:     next
16: end while
17: Output: The best solution found (bestSolution)
```

Figure 15. The standard pseudocode of all improvement heuristics.

For each method all feasible moves are examined and once an improvement is found the move is implemented, i.e. a first improvement approach is employed. Therefore, if a better solution is found, it becomes the incumbent solution and the algorithm is restarted. The heuristics end when a better solution cannot be found after exploring all feasible neighbourhood solutions.

In order to illustrate the application of each proposed move operator, an instance with 3 jobs $J = \{1, 2, 3\}$ and 5 resources $R = \{\mathrm{R1}, \mathrm{R2}, \mathrm{R3}, \mathrm{R4}, \mathrm{R5}\}$ will be used; a graphical representation of a feasible solution is shown in Figure 16. A solution is represented by a sequence of operations which appear according to the order in which they have been scheduled. Additionally, for each operation is also identified its assigned and alternative resources.

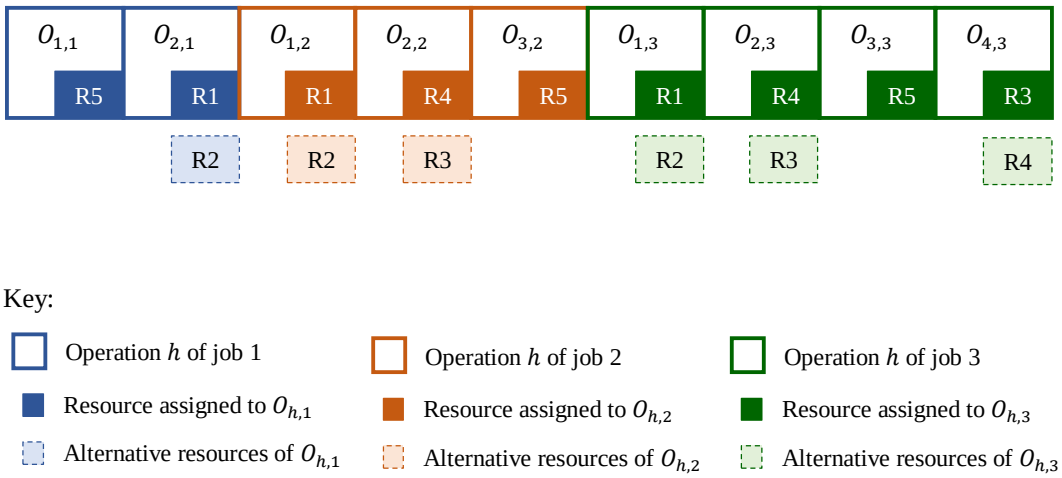

Figure 16. Graphical representation of a feasible solution for a 3x5 instance.

Resource R1 is a HWC, resources R2, R3 and R5 are finite capacity NWC, and resource R4 is an infinite capacity NWC. Concerning precedence constraints, these can be seen in Figure 17, (a), (b) and (c), respectively for job 1, job 2 and job 3. Additionally job 2 is also a precedence of operation $O_{4,3}$.

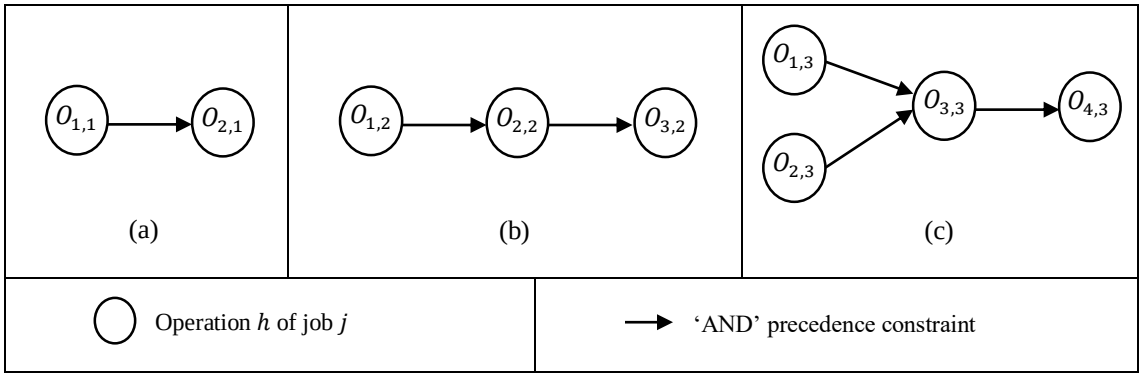

Figure 17. Precedence constraints of (a) job 1, (b) job 2 and (c) job 3.

The following subsections briefly describe and illustrate the move operator used in each of the proposed improvement heuristics. Move operators changing the assigned resources are grouped in subsection 4.2.1 and the ones changing operations sequence can be found in subsection 4.2.2. Finally, subsection 4.2.3 is composed of the move operator which simultaneously changes the assigned resources and operations sequence.

4.2.1. Changing Resources

Reassign. In the reassign improvement heuristic, an operation which has more than one eligible resource is selected and its current assigned resource is changed. Figure 18 illustrates a single move for a given selected operation ($O_{2,1}$).

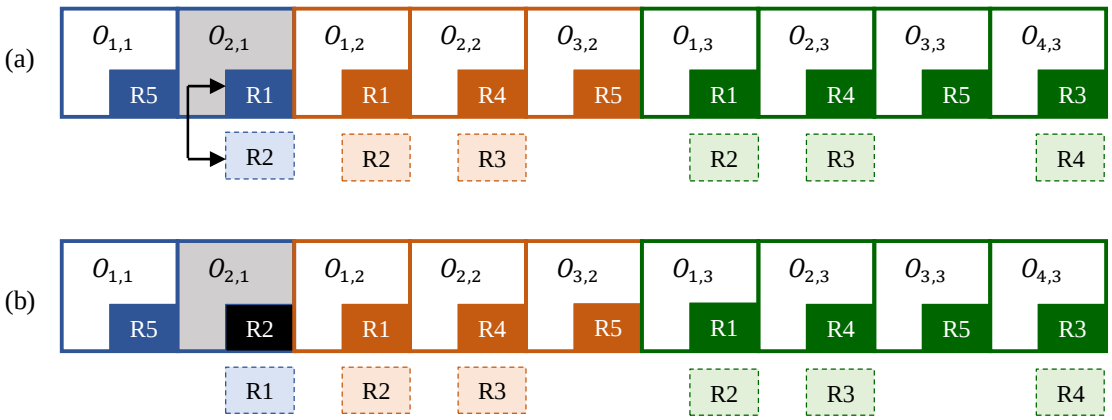

Figure 18. An example of a reassign move, (a) is the initial solution and (b) is a neighbourhood solution.

External Exchange. The external exchange improvement heuristic exchanges two operations among their current assigned resources. In each iteration it selects two operations and evaluates the feasibility of the exchange. An exchange is feasible if: (1) the resource currently assigned to one of the selected operations is also an eligible resource of the other one and vice versa; and (2) the two currently assigned resources are different. When conditions (1) and (2) are fulfilled, the two operations are exchanged and a neighbourhood solution is obtained. Figure 19 shows an example of a single move with operations $O_{2,2}$ and $O_{4,3}$.

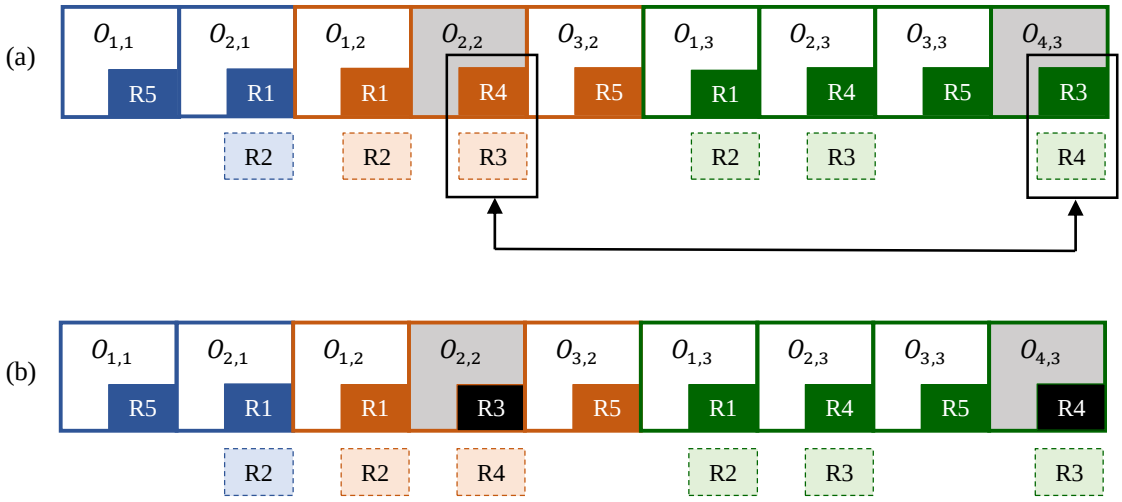

Figure 19. An example of an external exchange move, (a) is the initial solution and (b) is a neighbourhood solution.

4.2.2. Changing Operations Sequence

Internal Exchange. The internal exchange improvement heuristic exchanges the positions of two operations which are being performed in the same resource. Thus, it tries to modify the operations sequencing within resources.

For an exchange can be carry out two conditions have to be fulfilled: the two operations are not being processed by an infinite capacity NWC; and the exchange will not disrespect precedence constraints. In Figure 20 is represented a single move.

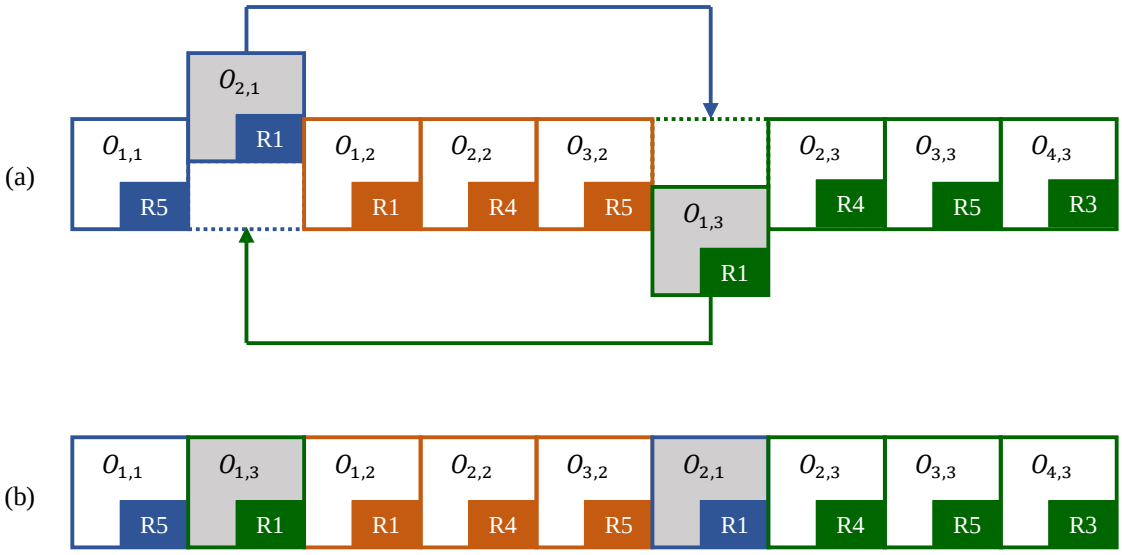

Figure 20. An example of an internal exchange move, (a) is the initial solution and (b) is a neighbourhood solution.

Swap. In the swap improvement heuristic an operation is selected and its position is exchanged with another operation in the sequence. This method is similar to previous one; the only difference is that the operations to be exchanged may not share the same resource. Therefore, it seeks to modify the operations sequencing both within and among resources.

4.2.3. Changing Resources and Operations Sequence

Reinsert-reassign. The reinsert-reassign improvement heuristic selects an operation, inserts it in another position in the operations sequence, and changes its assigned resource. An insertion is only carried out if no precedence constraints are violated and the selected operation has more than one eligible resource. Figure 21 shows all feasible insertions for a given operation ($O_{1,3}$) with the modification of its assigned resource.

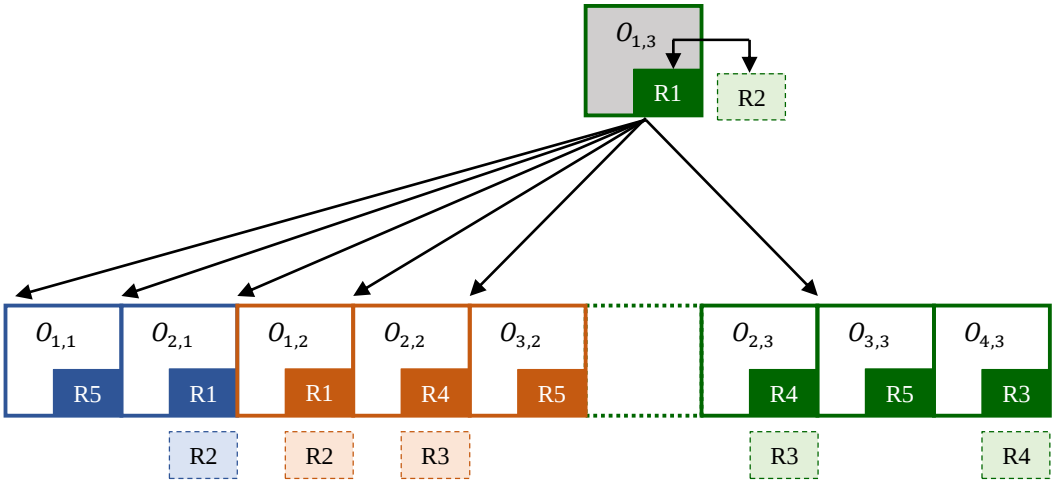

Figure 21. Possible insertions of a selected operation, with modification of its assigned resource.

4.3. Metaheuristics

This section details how two well-known metaheuristic approaches, GRASP and ILS, have been adapted for solving the previously described FJSP. These were chosen due to typically being among the most simple and efficient metaheuristics. Some of the previously developed constructive methods and improvement heuristics are employed in the proposed metaheuristics.

4.3.1. GRASP

GRASP is a multi-start metaheuristic in which each iteration consists of two phases, construction and local search (Resende and Ribeiro, 2003). The construction phase builds a feasible solution, and the local search is applied to found its local minimum. The final solution is the best solution found throughout the procedure.

Two variations of GRASP are proposed, based on two of the previously presented constructive heuristics: GRASP-JBJ and GRASP-OBO. The only difference between the two variations is the used constructive algorithm. Experimental test of GRASP using RBR showed significantly worse performances and therefore this possible variation was not further explored.

Both constructive methods were randomized in order to provide different starting solutions in the GRASP. Randomness in these constructive methods may be applied in two different stages: (1) in the selection of one candidate operation to be scheduled and (2) in the assignment of an eligible

resource to the selected operation. The former option was chosen over the latter and a combination of both, as experimental testing showed better results. Moreover, in JBJ all jobs are still randomly ordered before being ordered by precedences.

The local search is based on two improvement heuristics, applied sequentially, until no improvement can be found. The improvement heuristics complement each other as one changes the operations sequence and the other changes the assigned resources.

Figure 22 provides the pseudocode of the GRASP approach where it is the counter of successive iterations and $maxItGrasp$ is the maximum number of iterations.

```
1:  Input: J, R, maxItGrasp
2:  it ← 0; f(bestSolution) ← +∞
3:  while it < maxItGrasp
4:      solution_it ← ∅
5:      RandomizedConstructiveHeuristic(J, R, solution_it)
6:      LocalSearch(solution_it)
7:      if f(solution_it) < f(bestSolution)
8:          bestSolution ← solution_it
9:      end if
10:     it ← it + 1
11: end while
12: return bestSolution
```

Figure 22. Pseudocode of the GRASP approach.

4.3.2. ILS

ILS is a single-start metaheuristic, which starts from an initial solution and iteratively performs local search (intensification) and perturbation (diversification) phases. The latter phase aims overcoming getting trapped in local optima. It consists in changing local optimum solutions, obtained by local search, in order to direct the search to other regions of the solutions space.

Two versions of ILS are provided based on the constructive algorithm used: ILS-JBJ and ILS-OBO. In this case the constructive algorithms were not randomized – it has been shown that initial solutions provided by deterministic constructive algorithms often lead to better final results, and typically the local search requires less computation time (Lourenço et al., 2003).

The local search is performed similarly as in the GRASP approach, with two complementary improvement heuristics applied sequentially until no additional improvement can be found.

The perturbation function changes a local optimum solution in the following way: two positions of the operations sequence are randomly chosen and all operations among these two positions are reversed (sub-perturbation 1). However, the resulting solution may be unfeasible due to disrespecting precedence constraints and therefore it has to be reordered to become feasible (sub-perturbation 2). These two sub-perturbations are able to change a given solution in such way that the local search performed afterwards has less probability of falling into a previously visited local optimum.

The solution perturbed in each iteration is always the latter local optimum found by the local search (instead of the best solution found so far). The goal is to explore the highest possible number of neighbourhoods of the solutions space.

In Figure 23 the pseudocode of the proposed ILS algorithm is provided, where it is the counter of successive iterations.

```
1:  Input: J, R, maxItPert
2:  it ← 0; f(bestSolution) ← +∞; solution ← ∅
3:  ConstructiveHeuristic(J, R, solution)
4:  LocalSearch(solution)
5:  while it < maxItPert
6:      perturbedSol_it ← Perturbation(solution)
7:      solution ← LocalSearch(perturbedSol_it)
8:      if f(solution) < f(bestSolution) then
9:          bestSolution ← solution
10:     end if
11:     it ← it + 1
12: end while
13: return bestSolution
```

Figure 23. Pseudocode of the ILS approach.

The algorithm ends when the perturbation's maximum number of iterations ($maxItPert$) has been reached, returning the best solution found ($bestSolution$).

4. Heuristic Approaches

Chapter 5

Computational Evaluation

This section provides the computational evaluation which was carried out to assess the performance of the proposed methods. Firstly, implementation aspects are described and two sets of well-known benchmark instances from the general FJSP's literature are adapted to meet the specifications of the problem. Secondly, a comparative analysis is carried out using the newly proposed test instances.

5.1. Implementation

All the algorithms were coded in Visual Basic.NET and results obtained using a 2.30 GHz Intel Core i5-2410 CPU with 4 GB of RAM and Windows 10. Two sets of test instances, adapted from well-known benchmark instances of the FJSP literature, were used to evaluate the performance of the algorithms. These are the sets from Kacem et al. (2002) "*KacemData*" and from Brandimarte (1993) "*BRData*" with, respectively, 4 and 15 instances. Additionally, an instance based on real data from a Softi9's customer (with a project-oriented production system) was included. This instance does not include some problem's characteristics such as HWC resources, infinite capacity resources and sequence-dependent setup times.

The two sets of test instances had to be adapted as they are directed at evaluating the general version of FJSPs. In its general version, the objective is the minimisation of makespan and the following characteristics are considered:

- Jobs are independent from each other. There are no precedence constraints among the operations of different jobs. Nevertheless, there are linear precedence constraints among the operations of the same job;
- All jobs and resources are available at time zero;
- Setup times of resources are negligible;
- At a given time, a resource can only execute one operation;

- Resources are always available;
- The processing times of operations are constant and known in advance.

Therefore, the following data had to be defined. Data regarding setups (i.e. times and characteristics), resource types (i.e. finite/infinite NWC and HWC), resources calendar with working and stopping periods[9] (including changes in capacity), jobs due dates, operations release dates, precedence relations between different jobs, and non-linear precedence relations between operations of the same job.

Table 4 shows the main characteristics of the used test instances. In the first column is the instance name which is composed of two letters allowing identifying the author (e.g. "Km" for Kacem and "Mk" for Brandimarte) and a number (e.g. "Km01" is the instance number one of Kacem). A new real-world instance is also added: "R01". The following columns display the number of jobs (J), resources (R) and operations (O). The last two columns refer to the minimum and maximum number of operations per job (H_j) and eligible resources per operation (R_{hj}).

Table 4. Main characteristics of the test instances.

	Instance	$\|J\|$	$\|R\|$	$\|O\|$	H_j	$\|R_{h,j}\|$
1	Km01	4	5	12	[2, 4]	[1, 2]
2	Km02	10	7	29	[2, 3]	[1, 3]
3	Km03	10	10	30	3	[1, 3]
4	Km04	15	10	56	[2, 4]	[2, 3]
5	Mk01	10	6	55	[5, 6]	[1, 3]
6	Mk02	10	6	58	[5, 6]	[1, 3]
7	Mk03	15	8	150	10	[1, 3]
8	Mk04	15	8	90	[3, 9]	[1, 3]
9	Mk05	15	4	106	[5, 9]	[1, 2]
10	Mk06	10	10*	150	15	[1, 3]
11	Mk07	20	5	100	5	[1, 2]
12	Mk08	20	10	225	[10, 14]	[1, 2]
13	Mk09	20	10	240	[10, 14]	[1, 3]
14	Mk10	20	10*	240	[10, 14]	[1, 3]
15	Mk11	30	5	179	[5, 7]	[1, 2]
16	Mk12	30	10	193	[5, 9]	[1, 2]
17	Mk13	30	10	231	[5, 9]	[1, 3]
18	Mk14	30	15	277	[8, 11]	[1, 2]
19	Mk15	30	15	284	[8, 11]	[1, 3]
20	R01	111	40	388	[1, 15]	[1, 6]

* Although original instance has 15 resources, only 10 resources have data concerning to operations processing times.

[9] Aiming to simplify input data structure, the non-working periods were considered as stopping periods.

Preliminary testing allowed tuning the heuristics' parameters and determining the two improvement heuristics more suitable to be used in the local search.

The proposed GRASP and ILS have one main parameter, the maximum number of iterations, which was determined to depend on the size of the test instances. The size of the test instances is based on their total number of operations, $|O|$. It was considered that small, mid-size and large instances have, respectively, $|O| < 100$, $100 \leq |O| < 225$, and $|O| \geq 225$. Regarding GRASP, $maxItGrasp$ is 50 for smaller instances, 30 for mid-sized instances and 10 for larger instances. In ILS, $maxItPert$ takes on value 100 for smaller instances, 50 for mid-sized instances and 10 for larger instances. These parameters enable to obtain results in reasonable computing times, even for larger instances, ensuring both methods are given similar times.

The two improvement heuristics chosen to be used in the local search phase were the internal exchange and reassign. Choice is due to being the fastest methods, respectively, in changing the operations sequence and the assigned resources.

A hybrid metaheuristic approach combining features of the GRASP and the ILS approaches was also implemented and tested. However, preliminary results did not show improvement over the results by either the GRASP or the ILS approaches. Therefore, the method is disregarded.

5.2. Comparative Analysis

Results for the constructive methods, with and without improvement heuristics and local search (composed of internal exchange and reassign improvement heuristics), as well as for the metaheuristics approaches were obtained for all the test instances. Regarding constructive and improvement methods, for each instance, a single run was performed, from which was collected the objective function value, the makespan, and the computing time. The makespan value was collected in order to try to further understand how the schedules (solutions) are distributed along the scheduling horizon. Solutions with smaller makespan values could point to shorter resources idle times between consecutive operations, which may be valued by decision makers. For metaheuristic approaches, ten runs were made for each instance, from which were obtained objective function, makespan and computing time values.

Table 5 shows average and median results for the three proposed constructive methods with and without the improvement heuristics and local search. The first column identifies the method name and the following provide average and median values of CPU times (in seconds), gap to the best

results found during the experimentation phase (Gap_{BR}) and makespan. The detailed results for each instance can be found in Appendix B.

Table 5. Average and median results for the constructive methods with and without the improvement heuristics and local search.

Method	CPU (s)		Gap_{BR}		Makespan	
	Average	Median	Average	Median	Average	Median
JBJ	0.14	0.14	8.60	8.00	499.83	391.40
JBJ + Reassign	8.71	1.51	4.95	5.50	472.81	391.40
JBJ + External Exchange	31.88	4.62	6.45	4.50	494.41	387.45
JBJ + Internal Exchange	9.81	0.40	7.75	7.00	500.20	391.40
JBJ + Swap	176.18	2.60	7.05	6.50	495.08	391.40
JBJ + Reinsert-reassign	252.46	18.00	4.60	4.50	467.12	391.40
JBJ + Local search	22.66	2.94	5.10	6.00	474.91	391.40
OBO	0.43	0.18	5.15	5.00	411.06	313.65
OBO + Reassign	5.55	1.72	3.40	3.50	407.62	306.85
OBO + External Exchange	26.87	3.98	2.95	2.00	409.36	306.85
OBO + Internal Exchange	1.58	0.47	4.05	4.50	408.61	306.85
OBO + Swap	27.65	3.11	3.30	3.00	408.91	305.80
OBO + Reinsert-reassign	240.27	17.75	3.05	2.50	405.43	303.40
OBO + Local search	9.88	3.18	3.05	2.00	409.03	306.85
RBR	0.26	0.15	7.10	8.00	430.99	329.95
RBR + Reassign	8.10	1.38	4.90	5.00	427.74	313.60
RBR + External Exchange	33.48	4.80	4.30	4.00	428.41	330.50
RBR + Internal Exchange	2.87	0.43	6.20	7.00	430.32	330.75
RBR + Swap	63.76	8.58	5.20	5.00	429.43	330.95
RBR + Reinsert-reassign	253.25	14.18	4.80	5.00	425.97	331.70
RBR + Local search	16.95	3.37	4.60	4.50	429.32	330.75

Focusing on the constructive methods, results suggest that OBO outperforms JBJ and RBR regarding solution quality (the lowest average values of Gap_{BR} and makespan). However, JBJ and RBR require less CPU times. This is possibly due to the way resources are chosen for performing operations: in JBJ and RBR resource assignment is independent of operations starting times, while in OBO operation starting times in eligible resources is taken into account. The latter method obtains better results at the expense of more computation time.

Regarding results of improvement heuristics, these may lead to conclude that reassign and internal exchange are the fastest out of the methods changing resource assignment and operations sequence. However, the obtained improvements are on average smaller than the ones provided by the other similar methods. Within improvement heuristics intending changing assigned resources, external exchange usually obtains better gaps; concerning improvement methods changing operations

sequence, swap consistently outperforms the remaining. However, when internal exchange and reassign methods are employed in the local search they are able to reach the best trade-off relative to CPU time, gaps and makespan. The reinsert-reassign improvement heuristic presents the best average result of makespan (for all constructive heuristics) and competitive improvements but its larger CPU times makes it less interesting than the remaining improvement heuristics.

Results concerning metaheuristic approaches (GRASP-JBJ, GRASP-OBO, ILS-JBJ, and ILS-OBO) are shown in Table 6 and Table 7. In the tables, the first column displays the instance name and the second one the overall best result, BR. The following columns shown for each algorithm: the obtained results (average results, Avg., best results, Best, and average results of makespan, Avg. Mkp.); CPU time, in seconds; and the gap (Gap_{BR}) between the algorithm's best result and the overall best result. Moreover, in each of the tables average and median values for makespan (Avg. Mkp.), CPU time, and Gap_{BR} are provided.

Table 6. Results for the GRASP (using JBJ and OBO) metaheuristic approach.

			GRASP-JBJ					GRASP-OBO				
	Instance	BR	Avg.	Best	Avg. Mkp.	CPU (s)	Gap_{BR}	Avg.	Best	Avg. Mkp.	CPU (s)	Gap_{BR}
1	Km01	4	4.00	4	67.60	0.14	0	4.00	4	72.75	0.14	0
2	Km02	7	7.40	7	127.73	2.07	0	7.00	7	104.66	2.57	0
3	Km03	7	8.00	8	84.08	2.98	1	8.00	8	83.55	1.67	1
4	Km04	5	8.90	8	132.02	10.05	3	6.20	6	83.98	10.54	1
5	Mk01	2	4.10	4	73.85	8.03	2	2.00	2	41.00	8.87	0
6	Mk02	1	3.30	3	82.57	9.84	2	2.00	2	58.98	8.80	1
7	Mk03	6	9.50	9	422.21	111.53	3	6.80	6	252.93	117.92	0
8	Mk04	4	6.40	6	102.20	40.00	2	5.50	5	59.96	51.29	1
9	Mk05	11	11.90	11	324.22	23.73	0	12.40	12	299.72	35.54	1
10	Mk06	2	6.00	6	300.58	189.96	4	3.50	3	144.41	134.75	1
11	Mk07	10	12.00	10	301.89	20.83	0	11.10	10	290.42	43.47	0
12	Mk08	7	8.50	7	739.42	100.49	0	9.40	8	598.77	124.11	1
13	Mk09	5	8.30	7	570.50	117.33	2	9.20	8	515.88	170.76	3
14	Mk10	6	8.50	6	516.02	155.75	0	8.10	6	519.08	198.52	0
15	Mk11	13	16.50	15	895.48	173.76	2	18.70	18	869.53	450.70	5
16	Mk12	16	17.80	16	869.66	146.71	0	17.70	17	822.85	287.59	1
17	Mk13	16	17.80	17	784.42	180.14	1	19.70	18	624.22	214.31	2
18	Mk14	18	20.70	19	1181.79	235.84	1	21.30	19	902.01	293.00	1
19	Mk15	13	15.80	13	687.29	288.89	0	18.00	17	586.53	339.67	4
20	R01	6	11.33	10	900.50	8684.00	4	8.30	7	751.98	1625.09	1
				Average	458.20	525.10	1.35			384.16	205.97	1.20
				Median	373.22	106.01	1.00			295.07	121.02	1.00

Table 7. Results for the ILS (using JBJ and OBO) metaheuristic approach.

			ILS-JBJ					ILS-OBO				
	Instance	BR	Avg.	Best	Avg. Mkp.	CPU (s)	Gap_{BR}	Avg.	Best	Avg. Mkp.	CPU (s)	Gap_{BR}
1	Km01	4	4.00	4	67.60	0.27	0	4.00	4	64.50	0.22	0
2	Km02	7	8.00	7	169.29	3.07	0	8.00	8	180.99	2.75	1
3	Km03	7	7.10	7	81.72	2.69	0	7.40	7	70.83	2.33	0
4	Km04	5	6.50	5	114.92	13.23	0	5.70	5	145.55	10.67	0
5	Mk01	2	2.90	2	57.12	11.57	0	2.90	2	48.51	11.14	0
6	Mk02	1	2.20	2	73.90	14.07	1	1.90	1	60.49	12.62	0
7	Mk03	6	9.30	7	357.39	111.57	1	6.60	6	250.33	123.90	0
8	Mk04	4	5.90	5	82.00	52.54	1	5.00	4	79.87	57.37	0
9	Mk05	11	11.90	11	350.17	38.52	0	12.00	12	334.30	34.54	1
10	Mk06	2	5.70	5	209.67	153.69	3	2.80	2	149.86	191.41	0
11	Mk07	10	11.50	11	290.32	25.71	1	11.00	10	291.11	22.33	0
12	Mk08	7	10.10	8	728.55	72.63	1	7.90	7	647.91	77.59	0
13	Mk09	5	8.30	7	575.02	114.25	2	5.90	5	577.01	120.33	0
14	Mk10	6	9.30	7	524.55	143.82	1	7.70	6	518.78	136.04	0
15	Mk11	13	15.70	13	863.49	274.24	0	16.80	16	872.13	251.90	3
16	Mk12	16	18.40	17	859.12	112.38	1	16.80	16	820.52	199.63	0
17	Mk13	16	17.50	16	838.37	165.91	0	16.60	16	673.48	141.09	0
18	Mk14	18	22.30	20	1178.22	170.72	2	21.30	18	116.80	169.95	0
19	Mk15	13	18.40	17	742.03	199.88	4	15.40	14	619.69	229.73	1
20	R01	6	11.00	9	869.94	1647.32	3	6.50	6	618.73	860.99	0
				Average	451.67	166.40	1.05			357.07	132.83	0.30
				Median	353.78	92.10	1.00			270.72	98.96	0.00

Looking at results of metaheuristic approaches, ILS-OBO shows the best performance (the best average values of Gap_{BR} and CPU time), followed by ILS-JBJ, GRASP-OBO, and GRASP-JBJ. The latter three present average Gap_{BR} values around four times higher than the ones obtained by ILS-OBO.

The GRASP (either using JBJ or OBO) presents worse results when compared with ILS. This can be due to the diversification mechanism employed: in the GRASP it is associated with the randomization in the constructive algorithms, whereas in the ILS it is implemented through the perturbation method. This may lead to concluded that the perturbation method typically works better than the randomization in the search for better solutions, despite in smaller/medium-sized instances less iterations being performed in the GRASP.

Another conclusion to point out is that OBO constructive method seems to have better performance than JBJ when used in these metaheuristic approaches (GRASP and ILS). This is mostly due to the different ways resources are chosen for performing operations in each of the methods. Thus, solutions obtained by OBO may be better starting points to the intensification phase of both metaheuristic approaches (GRASP-OBO and ILS-OBO).

Finally, most solutions with smaller values of total number of tardy jobs also present smaller values of makespan. This may be due to operations starting times being closer to the start of the schedule horizon, meaning that operations wait less time to be scheduled and, consequently, smaller idle times between consecutive operations may exist.

Figure 24 shows a graphical representation, using a Gantt chart, of part of the best solution found for test instance “R01”. The vertical axis (on the left side) displays the available resources and the horizontal axis (on the top) represents scheduling time horizon. Furthermore, operations belonging to the same job share the same colour and tardy operations (of tardy jobs) are identified with small red marks over them.

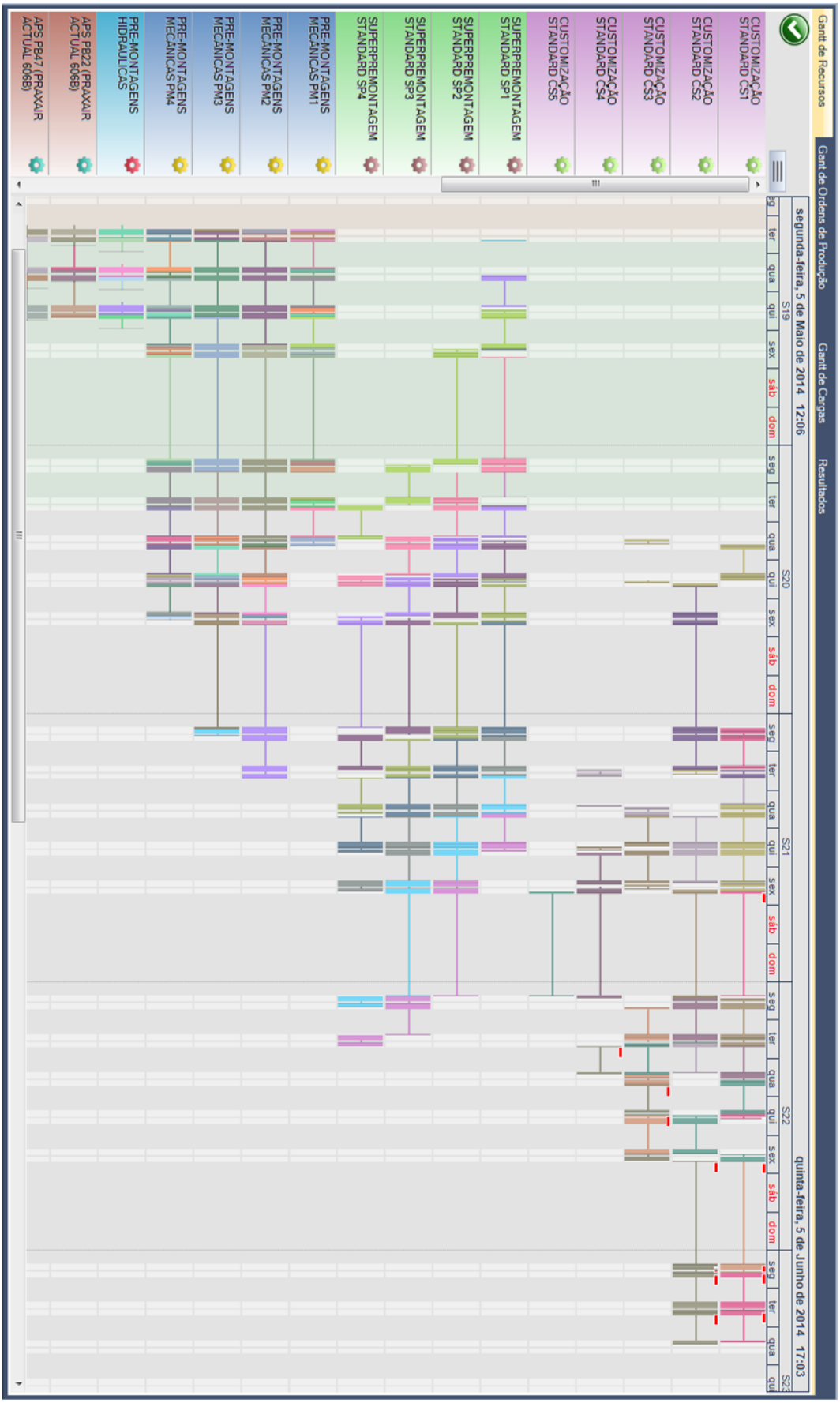

Figure 24. Gantt chart of the best solution found for instance "R01".

Chapter 6

Conclusions

In this dissertation a scheduling problem based on real-world industrial applications was addressed. Following a broadly accepted classification, concerning shop floor configuration, the problem can be classified as a FJSP. Moreover, several processing constraints such as release dates, precedence constraints, resources capacity (including downtimes and changing of capacity), and setup times are also considered. The objective of the problem is the minimisation of the total number of tardy jobs, which is considered the most important by the majority of schedulers (decision makers).

The literature review showed the FJSPs to be the closest to real-world production scenarios regarding shop floor configuration. However, the previously addressed FJSPs still don't address several important real-world processing characteristics which can be found in most scheduling scenarios. Additionally, scheduling problems are becoming increasingly complex due to industries having to produce an increasing diversity of products in smaller quantities driven by market demand. Therefore, the development of new methods to tackle specifications of these real scheduling problems is needed more than ever. As FJSPs are NP-hard, in order to solve real-life instances in reasonable times, heuristic approaches have been the main focus. Therefore, three constructive methods, five improvement heuristics and two metaheuristic approaches (using some of the previous methods) are proposed.

As the FJSP addressed herein has not been addressed in the literature, two sets of benchmark instances from the general FJSP literature were extended in order to computationally evaluate the proposed approaches. Among constructive methods, OBO presented the best performance and could be seen as an interesting method for schedulers whose main objective is to obtain solutions with some quality in a short time. The metaheuristic approach that performed best was the ILS with OBO constructive method (ILS-OBO). The approach showed competitive results in terms of total number of tardy jobs, makespan and computing times.

6.1. Limitations

Some limitations were identified during the development of this work.

The project's duration may be pointed out as the main limitation. Other important processing characteristics (e.g. sub-resources and time lags - maximal and minimal waiting times between different operations), also present in most production processes of Softi9's customers, had to be left out.

Being the first time that the described FJSP is addressed, there aren't other approaches to compare with, making harder the task of evaluating the proposed approaches and drawing conclusions on their performance. Moreover, lower bounds of the test instances were not obtained, possibly affecting the conclusions reached.

Another limitation has to do with the number of test instances used to evaluate the proposed heuristic approaches. Only nineteen randomly generated instances (adapted from the literature) and one real-life instance were used. Extending benchmark instances to fit the specific characteristics of the problem was considered a time-consuming and case-by-case analysis and, due to confidentiality reasons, real-life instances from Softi9's customers cannot be made available.

6.2. Future Work

Some future work opportunities and research directions were also identified throughout this work. The most relevant are addressed as follows.

The addressed FJSP handled some of the main processing characteristics found in many real-world manufacturing systems. Aiming to extend the applicability to other contexts it would be interesting to consider other processing characteristics such as sub-resources utilization, time lags, batch resources, among others.

Despite the minimisation of the total number of tardy jobs being the main objective for most schedulers, taking into account other objectives is often relevant. Some examples are the minimisation of average tardiness, waiting times and setup times, and the maximisation of resources workload and jobs net value. Multi-objective approaches may therefore be worth considering in future developments. This view is shared by Allahverdi (2015) which considers that there is a need for further works addressing multi-objective scheduling problems.

The proposed constructive and improvement heuristic approaches were developed without taking into consideration bottleneck resources, which could exist in a shop floor. This was due to *Softinov*

APS software still not being able to identify the bottleneck resources of a given scheduling problem. However, it is currently known the importance of optimizing production systems according to bottleneck resource(s), as they define the maximum output level. Thus, the development of new approaches or the adaptation existing ones focusing prioritizing bottleneck resources is a promising research avenue, with high applicability in companies which follow a scheduling approach based on the *Drum-Buffer-Rope* strategy (from Optimized Production Technology – OPT methodology) (see Goldratt, 1988).

Aiming to evaluate the applicability of the proposed algorithms to specific cases, and to further evaluate its performance, it would be useful to test them with a larger number and diversity of instances. It is therefore advisable the development of new sets of test instances, if possible based on real-world data. Moreover, as they would also allow to further fine-tune the parameters of the proposed approaches.

Finally it is worth noting the following: processing and shop floor characteristics found in real-world manufacturing companies are practically unlimited. Each scheduling problem in a given company has its own specifications making it virtually unique. Therefore, in order to better cope with the uniqueness of each production system, flexible solution-finding approaches should be envisioned, i.e. methods should be able to be easily tailored to cater for additional requirements efficiently.

6. Conclusions

References

Abdollahpour, S., & Rezaeian, J. (2015). Minimizing makespan for flow shop scheduling problem with intermediate buffers by using hybrid approach of artificial immune system. *Applied Soft Computing*, *28*(in press), 44–56. doi:10.1016/j.asoc.2014.11.022

Adams, J., Balas, E., & Zawack, D. (1988). The Shifting Bottleneck Procedure for job shop scheduling. *Management Science, 34*(3), 391-401.

Akkan, C. (2015). Improving schedule stability in single-machine rescheduling for new operation insertion. *Computers & Operations Research*, *64*(3797), 198–209. doi:10.1016/j.cor.2015.05.015

Al-Hinai, N., & ElMekkawy, T. Y. (2011). An efficient hybridized genetic algorithm architecture for the flexible job shop scheduling problem. *Flexible Services and Manufacturing Journal*, *23*(1), 64–85. doi:10.1007/s10696-010-9067-y

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, *246*(2), 345–378. doi:10.1016/j.ejor.2015.04.004

Allahverdi, A., & Aydilek, H. (2015). The two stage assembly flowshop scheduling problem to minimize total tardiness. *Journal of Intelligent Manufacturing*, *26*(2), 225–237. doi:10.1007/s10845-013-0775-5

Allahverdi, A., Ng, C. T., Cheng, T. C. E., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, *187*(3), 985–1032. doi:10.1016/j.ejor.2006.06.060

Almada-Lobo, B., Oliveira, J. F., & Carravilla, M. A. (2008). Production planning and scheduling in the glass container industry: A VNS approach. *International Journal of Production Economics*, *114*(1), 363–375. doi:10.1016/j.ijpe.2007.02.052

Alvarez-Valdes, R., Fuertes, A., Tamarit, J. M., Giménez, G., & Ramos, R. (2005). A heuristic to schedule flexible job-shop in a glass factory. *European Journal of Operational Research*, *165*(2), 525–534. doi:10.1016/j.ejor.2004.04.020

Amiri, M., Zandieh, M., Yazdani, M., & Bagheri, A. (2010). A variable neighbourhood search

algorithm for the flexible job-shop scheduling problem. *International Journal of Production Research*, *48*(19), 5671–5689. doi:10.1080/00207540903055743

Askin, R. G., & Standridge, C. R. (1993). *Modelling and Analysis of Manufacturing Systems*. New York, USA: John Wiley & Sons.

Avalos-Rosales, O., Angel-Bello, F., & Alvarez, A. (2015). Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, *76*(9-12), 1705–1718. doi:10.1007/s00170-014-6390-6

Azadeh, A., Farahani, M. H., Kalantari, S. S., & Zarrin, M. (2015). Solving a multi-objective open shop problem for multi-processors under preventive maintenance. *International Journal of Advanced Manufacturing Technology*, *78*(5-8), 707–722. doi:10.1007/s00170-014-6660-3

Bagheri, A., & Zandieh, M. (2011). Bi-criteria flexible job-shop scheduling with sequence-dependent setup times — Variable neighborhood search approach. *Journal of Manufacturing Systems*, *30*(1), 8–15. doi:10.1016/j.jmsy.2011.02.004

Bai, J., Li, Z., & Huang, X. (2012). Single-machine group scheduling with general deterioration and learning effects. *Applied Mathematical Modelling*, *36*(3), 1267–1274. doi:10.1016/j.apm.2011.07.068

Baker, K. R., & Trietsch, D. (2009). *Principles of Sequencing and Scheduling*. New Jersey, USA: John Wiley & Sons.

Balas, E., Simonetti, N., & Vazacopoulos, A. (2008). Job shop scheduling with setup times, deadlines and precedence constraints. *Journal of Scheduling*, *11*(4), 253–262. doi:10.1007/s10951-008-0067-7

Benavides, A. J., & Ritt, M. (2016). Two simple and effective heuristics for minimizing the makespan in non-permutation flow shops. *Computers & Operations Research*, *66*(3833), 160–169. doi:10.1016/j.cor.2015.08.001

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, *35*(3), 268–308. doi:10.1145/937503.937505

Bouras, A., Ghaleb, M. A., Suryahatmaja, U. S., & Salem, A. M. (2014). The Airport Gate Assignment Problem: A Survey. *The Scientific World Journal*, *2014*(923859), 1–27. doi: 10.1155/2014/923859

Boussaïd, I., Lepagnot, J., & Siarry, P. (2013). A survey on optimization metaheuristics. *Information*

Sciences, *237*(in press), 82–117. doi:10.1016/j.ins.2013.02.041

Bozorgirad, M. A., & Logendran, R. (2012). Sequence-dependent group scheduling problem on unrelated-parallel machines. *Expert Systems with Applications*, *39*(10), 9021–9030. doi:10.1016/j.eswa.2012.02.032

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, *41*(3), 157–183. doi:10.1007/BF02023073

Braune, R., & Zäpfel, G. (2016). Shifting bottleneck scheduling for total weighted tardiness minimization - A computational evaluation of subproblem and re-optimization heuristics. *Computers & Operations Research*, *66*(in press), 130–140. doi:10.1016/j.cor.2015.07.012

Brucker, P. (2007). *Scheduling Algorithms* (5th ed.). New York, USA: Springer.

Brucker, P., & Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing*, *45*(4), 369-375.

Buffa, E. S. & Sarin, R. K. (1987). *Modern Production/Operations Management*. New York, USA: John Wiley & Sons.

Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., & Qu, R. (2007). A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, *176*(1), 177–192. doi:10.1016/j.ejor.2005.08.012

Caballero-Villalobos, J. P., Mejía-Delgadillo, G. E., & García-Cáceres, R. G. (2013). Scheduling of complex manufacturing systems with Petri nets and genetic algorithms: a case on plastic injection moulds. *The International Journal of Advanced Manufacturing Technology*, *69*(9-12), 2773–2786. doi:10.1007/s00170-013-5175-7

Calleja, G., & Pastor, R. (2014). A dispatching algorithm for flexible job-shop scheduling with transfer batches: an industrial application. *Production Planning & Control*, *25*(2), 93–109. doi:10.1080/09537287.2013.782846

Chatavithee, P., Piewthongngam, K., & Pathumnakul, S. (2015). Scheduling a single machine with concurrent jobs for the frozen food industry. *Computers & Industrial Engineering*, *90*(in press), 158–166. doi:10.1016/j.cie.2015.09.004

Chen, H., Ihlow, J., & Lehmann, C. (1999, May). *A genetic algorithm for flexible job-shop scheduling*. Paper presented at the IEEE International Conference on Robotics and Automation, Detroit, *2*, 1120–1125.

Cheng, T. C. E., Hsu, C. J., Huang, Y. C., & Lee, W. C. (2011). Single-machine scheduling with

deteriorating jobs and setup times to minimize the maximum tardiness. *Computers & Operations Research*, *38*(12), 1760–1765. doi:10.1016/j.cor.2010.11.014

Choi, H. S., & Lee, D. H. (2009). Scheduling algorithms to minimize the number of tardy jobs in two-stage hybrid flow shops. *Computers & Industrial Engineering*, *56*(1), 113–120. doi:10.1016/j.cie.2008.04.005

Chuang, M., Liao, C., & Chao, C. (2010). Parallel machine scheduling with preference of machines. *International Journal of Production Research*, *48*(14), 4139–4152. doi:10.1080/00207540902991674

Conway, R. W., Maxwell, W. L., & Miller, L. W. (1967). *Theory of scheduling*. Massachusetts, USA: Addison-Wesley.

Cook, W. (2012). Markowitz and Manne + Eastman + Land and Doig = branch and bound. In Grötschel, M. (ed.), *Optimization Stories* (pp. 227–238). Bielefeld , Germany: Deutsche Mathematiker-Vereinigung.

Costa, A., Cappadonna, F. A., & Fichera, S. (2014). A novel genetic algorithm for the hybrid flow shop scheduling with parallel batching and eligibility constraints. *The International Journal of Advanced Manufacturing Technology*, *75*(5-8), 833–847. doi:10.1007/s00170-014-6195-7

Costa, W. E., Goldbarg, M. C., & Goldbarg, E. G. (2012). New VNS heuristic for total flowtime flowshop scheduling problem. *Expert Systems With Applications*, *39*(9), 8149–8161. doi:10.1016/j.eswa.2012.01.152

Creemers, S. (2015). Minimizing the expected makespan of a project with stochastic activity durations under resource constraints. *Journal of Scheduling*, *18*(3), 263–273. doi:10.1007/s10951-015-0421-5

Czibula, O. G., Gu, H., Hwang, F., Kovalyov, M. Y., & Zinder, Y. (2016). Bi-criteria sequencing of courses and formation of classes for a bottleneck classroom. *Computers & Operations Research*, *65*(3810), 53–63. doi:10.1016/j.cor.2015.06.010

Davidović, T., & Crainic, T. G. (2015). Parallel Local Search to schedule communicating tasks on identical processors. *Parallel Computing*, *48*(in press), 1–14. doi:10.1016/j.parco.2015.04.002

Demir, Y., & İşleyen, S. K. (2014). An effective genetic algorithm for flexible job-shop scheduling with overlapping in operations. *International Journal of Production Research*, *52*(13), 3905–3921. doi:10.1080/00207543.2014.889328

Dhouib, E., Loukil, T., & Teghem, J. (2013). Minimizing the Number of Tardy Jobs in a Permutation Flowshop Scheduling Problem With Setup Times and Time Lags Constraints. *Journal of*

Mathematical Modelling and Algorithms, *12*(1), 85–99. doi:10.1007/s10852-012-9180-x

Driessel, R., & Mönch, L. (2011). Variable neighborhood search approaches for scheduling jobs on parallel machines with sequence-dependent setup times , precedence constraints , and ready times. *Computers & Industrial Engineering*, *61*(2), 336–345. doi:10.1016/j.cie.2010.07.001

Elmi, A., Solimanpur, M., Topaloglu, S., & Elmi, A. (2011). A simulated annealing algorithm for the job shop cell scheduling problem with intercellular moves and reentrant parts. *Computers & Industrial Engineering*, *61*(1), 171–178. doi:10.1016/j.cie.2011.03.007

Fattahi, P., Jolai, F., & Arkat, J. (2009). Flexible job shop scheduling with overlapping in operations. *Applied Mathematical Modelling*, *33*(7), 3076–3087. doi:10.1016/j.apm.2008.10.029

Fernandez-viagas, V., & Framinan, J. M. (2015). Efficient non-population-based algorithms for the permutation flowshop scheduling problem with makespan minimisation subject to a maximum tardiness. *Computers & Operations Research*, *64*(in press), 86–96. doi:10.1016/j.cor.2015.05.006

Fondrevelle, J., Oulamara, A., & Portmann, M. C. (2008). Permutation flowshop scheduling problems with time lags to minimize the weighted sum of machine completion times. *International Journal of Production Economics*, *112*(1), 168–176. doi:10.1016/j.ijpe.2006.08.018

Framinan, J. M., Leisten, R., & García, R. R. (2014). *Manufacturing Scheduling Systems: An Integrated View on Models, Methods and Tools*. London, England: Springer.

Framinan, J. M., & Perez-Gonzalez, P. (2015). On heuristic solutions for the stochastic flowshop scheduling problem. *European Journal of Operational Research*, *246*(2), 413–420. doi:10.1016/j.ejor.2015.05.006

Gao, J., Gen, M., Sun, L., & Zhao, X. (2007). A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems. *Computers & Industrial Engineering*, *53*(1), 149–162. doi:10.1016/j.cie.2007.04.010

Gao, J., Sun, L., & Gen, M. (2008). A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research*, *35*(9), 2892–2907. doi:10.1016/j.cor.2007.01.001

Gao, K., Suganthan, P., Tasgetiren, M., Pan, Q., & Sun, Q. (2015). Effective ensembles of heuristics for scheduling flexible job shop problem with new job insertion. *Computers & Industrial Engineering*, *90*(4142), 107–117. doi:10.1016/j.cie.2015.09.005

Gao, K. Z., Suganthan, P. N., Chua, T. J., Chong, C. S., Cai, T. X., & Pan, Q. K. (2015). A two-stage

artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job insertion. *Expert Systems with Applications*, *42*(21), 7652–7663. doi:10.1016/j.eswa.2015.06.004

Gao, K. Z., Suganthan, P. N., Pan, Q. K., Chua, T. J., Cai, T. X., & Chong, C. S. (2016). Discrete harmony search algorithm for flexible job shop scheduling problem with multiple objectives. *Journal of Intelligent Manufacturing*, *27*(2), 363–374. doi:10.1007/s10845-014-0869-8

Gao, K. Z., Suganthan, P. N., Pan, Q. K., & Tasgetiren, M. F. (2015). An effective discrete harmony search algorithm for flexible job shop scheduling problem with fuzzy processing time. *International Journal of Production Research*, *53*(19), 5896–5911. doi:10.1080/00207543.2015.1020174

Gao, L., Li, X., Wen, X., Lu, C., & Wen, F. (2015). A hybrid algorithm based on a new neighborhood structure evaluation method for job shop scheduling problem. *Computers & Industrial Engineering*, *88*(in press), 417–429. doi:10.1016/j.cie.2015.08.002

Gao, Q., & Lu, X. (2014). Two-Machine Flow Shop Scheduling With Individual Operation's Rejection. *Asia-Pacific Journal of Operational Research*, *31*(01), 1–13. doi:10.1142/S021759591450002X

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, USA: W. H. Freeman and Company.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research, 1*(2), 117-129.

Gharbi, A., Ladhari, T., Msakni, M. K., & Serairi, M. (2013). The two-machine flowshop scheduling problem with sequence-independent setup times: New lower bounding strategies. *European Journal of Operational Research*, *231*(1), 69–78. doi:10.1016/j.ejor.2013.05.031

Gholami, O., & Sotskov, Y. N. (2014). Solving parallel machines job-shop scheduling problems by an adaptive algorithm. *International Journal of Production Research*, *52*(13), 3888–3904. doi:10.1080/00207543.2013.835498

Goldratt, E. (1988). Computerized shop floor scheduling. *International Journal of Production Research, 26*(3), 443-455.

González, M. A., Vela, C. R., & Varela, R. (2015). Scatter search with path relinking for the flexible job shop scheduling problem. *European Journal of Operational Research*, *245*(1), 35–45. doi:10.1016/j.ejor.2015.02.052

Gordon, V., Proth, J. M., & Chu, C. (2002). A survey of the state-of-the-art of common due date

assignment and scheduling research. *European Journal of Operational Research*, *139*(1), 1–25. doi:10.1016/S0377-2217(01)00181-3

Grabowski, J., & Pempera, J. (2000). Sequencing of jobs in some production system. *European Journal of Operational Research*, *125*(3), 535–550. doi:10.1016/S0377-2217(99)00224-6

Grabowski, J., & Pempera, J. (2007). The permutation flow shop problem with blocking. A tabu search approach. *Omega*, *35*(3), 302–311. doi:10.1016/j.omega.2005.07.004

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. H. G. R. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. *Annals of Discrete Mathematics*, *5*(C), 287-326. doi:10.1016/S0167-5060(08)70356-X

Gupta, J. N. D., & Stafford, E. F. (2006). Flowshop scheduling research after five decades. *European Journal of Operational Research*, *169*(3), 699–711. doi:10.1016/j.ejor.2005.02.001

Han, Y., Gong, D., Jin, Y., & Pan, Q. (2016). Evolutionary multi-objective blocking lot-streaming flow shop scheduling with interval processing time. *Applied Soft Computing*, *42*(in press), 229–245. doi:10.1016/j.asoc.2016.01.033

Herr, O., & Goel, A. (2016). Minimising total tardiness for a single machine scheduling problem with family setups and resource constraints. *European Journal of Operational Research*, *248*(1), 123–135. doi:10.1016/j.ejor.2015.07.001

Hoogeveen, H. (2005). Multicriteria scheduling. *European Journal of Operational Research*, *167*(3), 592–623. doi:10.1016/j.ejor.2004.07.011

Huang, J., & Süer, G. A. (2014). A Dispatching Rule-based Genetic Algorithm for Multi-Objective Job Shop Scheduling using Fuzzy Satisfaction Levels. *Computers & Industrial Engineering*, *86*(in press), 29–42. doi:10.1016/j.cie.2014.12.001

Huang, R. H., Yang, C. L., & Huang, Y. C. (2009). No-wait two-stage multiprocessor flow shop scheduling with unit setup. *International Journal of Advanced Manufacturing Technology*, *44*(9-10), 921–927. doi:10.1007/s00170-008-1865-y

Ishikawa, S., Kubota, R., & Horio, K. (2015). Effective hierarchical optimization by a hierarchical multi-space competitive genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, *42*(24), 9434–9440. doi:10.1016/j.eswa.2015.08.003

Iturriaga, S., Nesmachnow, S., Luna, F., & Alba, E. (2015). A parallel local search in CPU/GPU for scheduling independent tasks on large heterogeneous computing systems. *The Journal of Supercomputing*, *71*(2), 648–672. doi:10.1007/s11227-014-1315-6

References

Jayamohan, M. S., & Rajendran, C. (2000). New dispatching rules for shop scheduling: A step forward. *International Journal of Production Research*, *38*(3), 563–586. doi:10.1080/002075400189301

Jayamohan, M. S., & Rajendran, C. (2004). Development and analysis of cost-based dispatching rules for job shop scheduling. *European Journal of Operational Research*, *157*(2), 307–321. doi:10.1016/S0377-2217(03)00204-2

Jeong, B., & Kim, Y. D. (2014). Minimizing total tardiness in a two-machine re-entrant flowshop with sequence-dependent setup times. *Computers & Operations Research*, *47*(in press), 72–80. doi:10.1016/j.cor.2014.02.002

Jia, S., & Hu, Z. (2014). Path-relinking Tabu search for the multi-objective flexible job shop scheduling problem. *Computers & Operations Research*, *47*(in press), 11–26. doi:10.1016/j.cor.2014.01.010

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, *1*, 61–68.

Jolai, F., Sheikh, S., Rabbani, M., & Karimi, B. (2009). A genetic algorithm for solving no-wait flexible flow lines with due window and job rejection. *International Journal of Advanced Manufacturing Technology*, *42*(5-6), 523–532. doi:10.1007/s00170-008-1618-y

Kacem, I., Hammadi, S., & Borne, P. (2002). Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and Computers in Simulation*, *60*(3-5), 245–276. doi:10.1016/S0378-4754(02)00019-8

Keshavarz, T., Salmasi, N., & Varmazyar, M. (2015). Minimizing total completion time in the flexible flowshop sequence-dependent group scheduling problem. *Annals of Operations Research*, *226*(1), 351–377. doi:10.1007/s10479-014-1667-6

Kirlik, G., & Oguz, C. (2012). A variable neighborhood search for minimizing total weighted tardiness with sequence dependent setup times on a single machine. *Computers & Operations Research*, *39*(7), 1506–1520. doi:10.1016/j.cor.2011.08.022

Koulamas, C., & Panwalkar, S. S. (2015). On the equivalence of single machine earliness/tardiness problems with job rejection. *Computers & Industrial Engineering*, *87*(in press), 1–3. doi:10.1016/j.cie.2015.04.014

Kuhpfahl, J., & Bierwirth, C. (2016). A study on local search neighborhoods for the job shop scheduling problem with total weighted tardiness objective. *Computers & Operations Research*, *66*(in press), 44–57. doi:10.1016/j.cor.2015.07.011

Lawler, E. L., Lenstra, J. K., Kan, A. H. G. R., & Shmoys, D. B. (1993). Sequencing and Scheduling: Algorithms and Complexity. In S. . Graves, A. H. G. Rinnooy Kan, & P. H. Zipkin (Eds.), *Handbooks in Operations Research and Management Science* (pp. 445–522). Amesterdam, The Netherlands: Elsevier B.V.

Lee, S., Moon, I., Bae, H., & Kim, J. (2012). Flexible job-shop scheduling problems with "AND"/"OR" precedence constraints. *International Journal of Production Research, 50*(7), 1971-2001. doi:10.1080/00207543.2011.561375

Lei, D., & Guo, X. (2014). Variable neighbourhood search for dual-resource constrained flexible job shop scheduling. *International Journal of Production Research*, *52*(9), 2519–2529. doi:10.1080/00207543.2013.849822

Lei, D., & Guo, X. (2016). Hybrid flow shop scheduling with not-all-machines options via local search with controlled deterioration. *Computers & Operations Research*, *65*(3792), 76–82. doi:10.1016/j.cor.2015.05.010

Lei, D., & Guo, X. P. (2011). Variable neighbourhood search for minimising tardiness objectives on flow shop with batch processing machines. *International Journal of Production Research*, *49*(2), 519–529. doi:10.1080/00207540903536130

Li, J., Pan, Q., & Liang, Y. (2010). An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering*, *59*(4), 647–662. doi:10.1016/j.cie.2010.07.014

Li, J., Pan, Q., & Tasgetiren, M. F. (2014). A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities. *Applied Mathematical Modelling*, *38*(3), 1111–1132. doi:10.1016/j.apm.2013.07.038

Li, K., Jia, Z. H., & Leung, J. Y. (2015). Integrated production and delivery on parallel batching machines. *European Journal of Operational Research*, *247*(3), 755–763. doi:10.1016/j.ejor.2015.06.051

Li, X., Chen, H., Du, B., & Tan, Q. (2013). Heuristics to schedule uniform parallel batch processing machines with dynamic job arrivals. *International Journal of Computer Integrated Manufacturing, 26*(5), 474–486. doi:10.1080/0951192X.2012.731612

Liao, C., Tjandradjaja, E., & Chung, T. (2012). An approach using particle swarm optimization and bottleneck heuristic to solve hybrid flow shop scheduling problem. *Applied Soft Computing, 12*(6), 1755–1764. doi:10.1016/j.asoc.2012.01.011

Liu, Z., Xiao, L., & Tian, J. (2015). An activity-list-based nested partitions algorithm for resource-

constrained project scheduling. *International Journal of Production Research, in press*. doi:10.1080/00207543.2015.1065353

Lourenço, H. R., Martin, O. C., & Stützle, T. (2003). Iterated local search. In F. Glover, & G. A. Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 321–353). Dordrecht, The Netherlands: Kluwer Academic Publishers.

Luo, H., Zhang, A., & Huang, G. Q. (2012). Hybrid flowshop scheduling with family setup time and inconsistent family formation. *Journal of Intelligent Manufacturing*, *50*(6), 1457–1475. doi:10.1007/s10845-013-0771-9

Martí, R., & Reinelt, G. (2011). Heuristic Methods. In S. S. Antman, J. E. Marsden, & L. Sirovich (Eds.), *The Linear Ordering Problem, Exact and Heuristic Methods in Combinatorial Optimization* (Vol. 175, pp. 17–40). Berlim, Germany: Springer.

Mason, S. J., Fowler, J. W., & Carlyle, W. M. (2002). A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops. *Journal of Scheduling*, *5*(3), 247–262.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*(11), 1097–1100. doi:10.1016/S0305-0548(97)00031-2

Molina-sánchez, L. P., & González-Neira, E. M. (2016). GRASP to minimize total weighted tardiness in a permutation flow shop environment. *International Journal of Industrial Engineering Computations*, *7*(1), 161–176. doi:10.5267/j.ijiec.2015.6.004

Moore, J. M. (1968). A n job, one machine algorithm for minimizing the number of late jobs. *Management Science*, *15*(1), 102–109.

Moursli, O., & Pochet, Y. (2000). A branch-and-bound algorithm for the hybrid flowshop. *International Journal of Production Economics*, *64*(1), 113–125. doi:10.1016/S0925-5273(99)00051-1

Mousakhani, M. (2013). Sequence-dependent setup time flexible job shop scheduling problem to minimise total tardiness. *International Journal of Production Research*, *51*(12), 3476–3487. doi:10.1080/00207543.2012.746480

Murty, K. G. (2003). Heuristic Methods for Combinatorial Optimization Problems. In *Junior Level Web-Book for Optimization Models for decision Making* (pp. 425–508). Retrieved 17-11-2015, from http://ioe.engin.umich.edu/people/fac/books/murty/opti_model/junior-9.pdf

Naderi, B., Zandieh, M., Khaleghi Ghoshe Balagh, A., & Roshanaei, V. (2009). An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation

times to minimize total completion time and total tardiness. *Expert Systems with Applications*, *36*(6), 9625–9633. doi:10.1016/j.eswa.2008.09.063

Navaei, J., Fatemi Ghomi, S. M. T., Jolai, F., & Mozdgir, A. (2014). Heuristics for an assembly flow-shop with non-identical assembly machines and sequence dependent setup times to minimize sum of holding and delay costs. *Computers & Operations Research*, *44*(in press), 52–65. doi:10.1016/j.cor.2013.10.008

Nawaz, M., Enscore, E. E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, *11*(1), 91–95.

Nejati, M., Mahdavi, I., Hassanzadeh, R., Mahdavi-Amiri, N., & Mojarad, M. (2014). Multi-job lot streaming to minimize the weighted completion time in a hybrid flow shop scheduling problem with work shift constraint. *International Journal of Advanced Manufacturing Technology*, *70*(1-4), 501–514. doi:10.1007/s00170-013-5265-6

Neto, R. F. T., & Filho, M. G. (2011). An ant colony optimization approach to a permutational flowshop scheduling problem with outsourcing allowed. *Computers & Operations Research*, *38*(9), 1286–1293. doi:10.1016/j.cor.2010.11.010

Pan, Q. K., & Ruiz, R. (2013). A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Computers & Operations Research*, *40*(1), 117–128. doi:10.1016/j.cor.2012.05.018

Pang, K. W. (2013). A genetic algorithm based heuristic for two machine no-wait flowshop scheduling problems with class setup times that minimizes maximum lateness. *International Journal of Production Economics*, *141*(1), 127–136. doi:10.1016/j.ijpe.2012.06.017

Panwalkar, S. S., & Koulamas, C. (2014). The two-machine no-wait general and proportionate open shop makespan problem. *European Journal of Operational Research*, *238*(2), 471–475. doi:10.1016/j.ejor.2014.04.030

Papadimitriou, C. H., & Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*. New York, USA: Prentice-Hall.

Peng, B., Lü, Z., & Cheng, T. C. E. (2015). A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Computers & Operations Research*, *53*(in press), 154–164. doi:10.1016/j.cor.2014.08.006

Pezzella, F., Morganti, G., & Ciaschetti, G. (2008). A genetic algorithm for the Flexible Job-shop Scheduling Problem. *Computers & Operations Research*, *35*(10), 3202–3212. doi:10.1016/j.cor.2007.02.014

References

Pinedo, M. L. (2009). *Planning and Scheduling in Manufacturing and Services* (4th ed.). New York, USA: Springer.

Pinedo, M. L. (2012). *Scheduling: Theory, Algorithms, and Systems*. New York, USA: Springer.

Pinedo, M., & Singer, M. (1999). A Shifting Bottleneck Heuristic for Minimizing the Total Weighted Tardiness in a Job Shop. *Naval Research Logistics*, *46*(1), 1–17. doi:10.1002/(SICI)1520-6750(199902)46

Potts, C. N., & Kovalyov, M. Y. (2000). Scheduling with batching: a review. *European Journal of Operational Research*, *120*(2), 228–249. doi:10.1016/S0377-2217(99)00153-8

Potts, C. N., Sevast'janov, S. V., Strusevich, V. A., Van Wassenhove, L. N., & Zwaneveld, C. M. (1995). The Two-stage Assembly Scheduling Problem: Complexity and Approximation. *Operations Research*, *43*(2), 346–355.

Qing-dao-er-ji, R., & Wang, Y. (2015). A new hybrid genetic algorithm for job shop scheduling problem. *Computers & Industrial Engineering*, *88*(10), 273–283. doi:10.1016/j.cor.2011.12.005

Rahimi-Vahed, A. R., Javadi, B., Rabbani, M., & Tavakkoli-Moghaddam, R. (2008). A multi-objective scatter search for a bi-criteria no-wait flow shop scheduling problem. *Engineering Optimization*, *40*(4), 331–346. doi:10.1080/03052150701732509

Rahnamayan, S., Tizhoosh, H. R., & Salama, M. M. A. (2007). A novel population initialization method for accelerating evolutionary algorithms. *Computers & Mathematics with Applications*, *53*(10), 1605–1614. doi:10.1016/j.camwa.2006.07.013

Rajkumar, M., Asokan, P., Anilkumar, N., & Page, T. (2011). A GRASP algorithm for flexible job-shop scheduling problem with limited resource constraints. *International Journal of Production Research*, *49*(8), 2409–2423. doi:10.1080/00207541003709544

Ramezani, P., Rabiee, M., & Jolai, F. (2015). No-wait flexible flowshop with uniform parallel machines and sequence-dependent setup time: a hybrid meta-heuristic approach. *Journal of Intelligent Manufacturing*, *26*(4), 731–744. doi:10.1007/s10845-013-0830-2

Ramudhin, A., & Marier, P. (1996). The generalized Shifting Bottleneck Procedure. *European Journal of Operational Research*, *93*(1), 34–48. doi:10.1016/0377-2217(95)00135-2

Resende, M. G. C., & Ribeiro, C. C. (2003). Greedy Randomized Adaptive Search Procedures. In F. Glover, & G. A. Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 219–249). Dordrecht, The Netherlands: Kluwer Academic Publishers.

Riise, A., Mannino, C., & Burke, E. K. (2016). Modelling and solving generalised operational surgery scheduling problems. *Computers & Operations Research*, *66*(in press), 1–11. doi:10.1016/j.cor.2015.07.003

Rodrigues, R. D. F., Dourado, M. C., & Szwarcfiter, J. L. (2014). Scheduling problem with multi-purpose parallel machines. *Discrete Applied Mathematics*, *164*(PART 1), 313–319. doi:10.1016/j.dam.2011.11.033

Ronconi, D. P. (2004). A note on constructive heuristics for the flowshop problem with blocking. *International Journal of Production Economics*, *87*(1), 39–48. doi:10.1016/S0925-5273(03)00065-3

Roshanaei, V., Naderi, B., Jolai, F., & Khalili, M. (2009). A variable neighborhood search for job shop scheduling with set-up times to minimize makespan. *Future Generation Computer Systems*, *25*(6), 654–661. doi:10.1016/j.future.2009.01.004

Rossi, A. (2014). Flexible job shop scheduling with sequence-dependent setup and transportation times by ant colony with reinforced pheromone relationships. *International Journal of Production Economics*, *153*(in press), 253–267. doi:10.1016/j.ijpe.2014.03.006

Ruiz, R., Maroto, C., & Alcaraz, J. (2006). Two new robust genetic algorithms for the flowshop scheduling problem. *Omega*, *34*(5), 461–476. doi:10.1016/j.omega.2004.12.006

Ruiz, R., Şerifoğlu, F. S., & Urlings, T. (2008). Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research*, *35*(4), 1151–1175. doi:10.1016/j.cor.2006.07.014

Ruiz, R., & Stützle, T. (2008). An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, *187*(3), 1143–1159. doi:10.1016/j.ejor.2006.07.029

Ruiz, R., & Vázquez-Rodríguez, J. A. (2010). The hybrid flow shop scheduling problem. *European Journal of Operational Research*, *205*(1), 1–18. doi:10.1016/j.ejor.2009.09.024

Saidi-Mehrabad, M., Dehnavi-Arani, S., Evazabadian, F., & Mahmoodian, V. (2015). An Ant Colony Algorithm (ACA) for solving the new integrated model of job shop scheduling and conflict-free routing of AGVs. *Computers & Industrial Engineering*, *86*(in press), 2–13. http://doi.org/10.1016/j.cie.2015.01.003

Saidi-Mehrabad, M., & Fattahi, P. (2007). Flexible job shop scheduling with tabu search algorithms. *The International Journal of Advanced Manufacturing Technology*, *32*(5-6), 563–570. doi:10.1007/s00170-005-0375-4

References

Schaller, J. E. (2014). Minimizing total tardiness for scheduling identical parallel machines with family setups. *Computers & Industrial Engineering*, *72*(1), 274–281. doi:10.1016/j.cie.2014.04.001

Seidgar, H., Abedi, M., Tadayonirad, S., & Fazlollahtabar, H. (2015). A hybrid particle swarm optimisation for scheduling just-in-time single machine with preemption, machine idle time and unequal release times. *International Journal of Production Research*, *53*(6), 1912–1935. doi:10.1080/00207543.2014.970705

Sels, V., Gheysen, N., & Vanhoucke, M. (2012). A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions. *International Journal of Production Research, 50*(15), 4255–4270.

Senthilkumar, P. (2010). Literature Review of Single Machine Scheduling Problem with Uniform Parallel Machines. *Intelligent Information Management*, *02*(08), 457–474. doi:10.4236/iim.2010.28056

Sha, D. Y., & Hsu, C. (2008). A new particle swarm optimization for the open shop scheduling problem. *Computers & Operations Research*, *35*(10), 3243–3261. doi:10.1016/j.cor.2007.02.019

Shahsavari-Pour, N., & Ghasemishabankareh, B. (2013). A novel hybrid meta-heuristic algorithm for solving multi objective flexible job shop scheduling. *Journal of Manufacturing Systems*, *32*(4), 771–780. doi:10.1016/j.jmsy.2013.04.015

Shao, X., Liu, W., Liu, Q., & Zhang, C. (2013). Hybrid discrete particle swarm optimization for multi-objective flexible job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, *67*(9-12), 2885–2901. doi:10.1007/s00170-012-4701-3

Shin, H. J. (2015). A dispatching algorithm considering process quality and due dates: an application for re-entrant production lines. *The International Journal of Advanced Manufacturing Technology*, *77*(1-4), 249–259. doi:10.1007/s00170-014-6436-9

Shingo, S. (1985). *A Revolution in Manufacturing: The SMED System*. Portland, Oregon: Productivity Press.

Shoaardebili, N., & Fattahi, P. (2015). Multi-objective meta-heuristics to solve three-stage assembly flow shop scheduling problem with machine availability constraints. *International Journal of Production Research*, *53*(3), 944–968. doi:10.1080/00207543.2014.948575

Stevenson, W. J. (2005). *Operations Management*. Boston, USA: McGraw-Hill.

Subramanian, A., Battarra, M., & Potts, C. N. (2014). An Iterated Local Search heuristic for the

single machine total weighted tardiness scheduling problem with sequence-dependent setup times. *International Journal of Production Research*, *52*(9), 2729–2742. doi:10.1080/00207543.2014.883472

Süer, G. A., Ates, O. K., & Mese, E. M. (2014). Cell loading and family scheduling for jobs with individual due dates to minimise maximum tardiness. *International Journal of Production Research*, *52*(19), 5656–5674. doi:10.1080/00207543.2014.903343

Sun, L., Liu, W., Xu, B., & Chai, T. (2010). The scheduling of steel-making and continuous casting process using branch and cut method via CPLEX optimization. In *5th International Conference on Computer Sciences and Convergence Information Technology* (pp. 716–721). doi:10.1109/ICCIT.2010.5711147

T'kindt, V., & Billaut, J. (2006). *Multicriteria Scheduling: Theory, Models and Algorithms* (2nd ed.). Berlin, Germany: Springer.

Tadayon, B., & Salmasi, N. (2012). A two-criteria objective function flexible flowshop scheduling problem with machine eligibility constraint. *The International Journal of Advanced Manufacturing Technology*, *64*(5-8), 1001–1015. doi:10.1007/s00170-012-4052-0

Talbi, E. G. (2009). *Metaheuristics: From Design to Implementation*. New Jersey, USA: John Wiley & Sons.

Tang, L., & Liu, G. (2007). A mathematical programming model and solution for scheduling production orders in Shanghai Baoshan Iron and Steel Complex. *European Journal of Operational Research*, *182*(3), 1453–1468. doi:10.1016/j.ejor.2006.09.090

Tasgetiren, M. F., Liang, Y. C., Sevkli, M., & Gencyilmaz, G. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, *177*(3), 1930–1947. doi:10.1016/j.ejor.2005.12.024

Tay, J. C., & Ho, N. B. (2008). Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, *54*(3), 453–473. doi:10.1016/j.cie.2007.08.008

Tian, Y., Liu, D., Yuan, D., & Wang, K. (2013). A discrete PSO for two-stage assembly scheduling problem. *International Journal of Advanced Manufacturing Technology*, *66*(1-4), 481–499. doi:10.1007/s00170-012-4343-5

Trietsch, D. (1992). *Some notes on the application of Single Minute Exchange of Die (SMED)* (Report No. NPS-AS-92-019). Retrieved 30-11-2015 from

http://calhoun.nps.edu/bitstream/handle/10945/29513/somenotesonappli00trie.pdf?sequence=1

Türkyılmaz, A., & Bulkan, S. (2015). A hybrid algorithm for total tardiness minimisation in flexible job shop: genetic algorithm with parallel VNS execution. *International Journal of Production Research*, *53*(6), 1832–1848. doi:10.1080/00207543.2014.962113

Varmazyar, M., & Salmasi, N. (2012). Sequence-dependent flow shop scheduling problem minimising the number of tardy jobs. *International Journal of Production Research*, *50*(20), 5843–5858. doi:10.1080/00207543.2011.632385

Vilcot, G., & Billaut, J. (2011). A tabu search algorithm for solving a multicriteria flexible job shop scheduling problem. *International Journal of Production Research*, *49*(23), 6963–6980. doi:10.1080/00207543.2010.526016

Wang, J., Luh, P. B., Zhao, X., & Wang, J. (1997). An optimization-based algorithm for job shop scheduling. *Sadhana*, *22*(2), 241–256. doi:10.1007/BF02744491

Wang, L., Jin, J., Wang, J.-B., & Ji, P. (2015). Research on scheduling problems with general effects of deterioration and learning. *Information Sciences*, *307*(in press), 89–94. doi:10.1016/j.ins.2015.02.021

Wang, L., & Zheng, D. (2001). An effective hybrid optimization strategy for job-shop scheduling problems. *Computers & Operations Research*, *28*(6), 585–596. doi:10.1016/S0305-0548(99)00137-9

Wang, L., Zhou, G., Xu, Y., Wang, S., & Liu, M. (2012). An effective artificial bee colony algorithm for the flexible job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, *60*(1-4), 303–315. doi:10.1007/s00170-011-3610-1

Wang, S., Su, H., & Wan, G. (2015). Resource-constrained machine scheduling with machine eligibility restriction and its applications to surgical operations scheduling. *Journal of Combinatorial Optimization*, *30*(4), 982-995. doi:10.1007/s10878-015-9860-3

Wang, X., Gao, L., Zhang, C., & Shao, X. (2010). A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, *51*(5-8), 757–767. doi:10.1007/s00170-010-2642-2

Xia, W., & Wu, Z. (2005). An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering*, *48*(2), 409–425. doi:10.1016/j.cie.2005.01.018

Yagmahan, B., & Yenisey, M. M. (2008). Ant colony optimization for multi-objective flow shop scheduling problem. *Computers & Industrial Engineering*, *54*(3), 411–420. doi:10.1016/j.cie.2007.08.003

Yazdani, M., Amiri, M., & Zandieh, M. (2010). Flexible job-shop scheduling with parallel variable neighborhood search algorithm. *Expert Systems With Applications*, *37*(1), 678–687. doi:10.1016/j.eswa.2009.06.007

Zabihzadeh, S. S., & Rezaeian, J. (2016). Two meta-heuristic algorithms for flexible flow shop scheduling problem with robotic transportation and release time. *Applied Soft Computing Journal*, *40*(in press), 319–330. doi:10.1016/j.asoc.2015.11.008

Zammori, F., Braglia, M., & Castellano, D. (2014). Harmony search algorithm for single-machine scheduling problem with planned maintenance. *Computers & Industrial Engineering*, *76*(in press), 333–346. doi:10.1016/j.cie.2014.08.001

Zhang, C., Li, P., Guan, Z., & Rao, Y. (2007). A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, *34*(11), 3229–3242. doi:10.1016/j.cor.2005.12.002

Zhang, C. Y., Li, P., Rao, Y., & Guan, Z. (2008). A very fast TS/SA algorithm for the job shop scheduling problem. *Computers & Operations Research*, *35*(1), 282–294. doi:10.1016/j.cor.2006.02.024

Zhang, G., Gao, L., & Shi, Y. (2011). An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, *38*(4), 3563–3573. doi:10.1016/j.eswa.2010.08.145

Zhang, G., Shao, X., Li, P., & Gao, L. (2009). An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. *Computers & Industrial Engineering*, *56*(4), 1309–1318. doi:10.1016/j.cie.2008.07.021

Zhang, J., & Yang, J. (2016). Flexible job-shop scheduling with flexible workdays, preemption, overlapping in operations and satisfaction criteria : an industrial application. *International Journal of Production Research*, in press, 1-25. doi:10.1080/00207543.2015.1134839

Zhang, R., Song, S., & Wu, C. (2012). A two-stage hybrid particle swarm optimization algorithm for the stochastic job shop scheduling problem. *Knowledge-Based Systems*, *27*(in press), 393–406. doi:10.1016/j.knosys.2011.11.018

Zhang, S., & Gu, X. (2015). An effective discrete artificial bee colony algorithm for flow shop scheduling problem with intermediate buffers. *Journal of Central South University*, *22*(9),

3471–3484. doi:10.1007/s11771-015-2887-x

Zhou, R., Nee, A. Y. C., & Lee, H. P. (2009). Performance of an ant colony optimisation algorithm in dynamic job shop scheduling problems. *International Journal of Production Research*, *47*(11), 2903–2920. doi:10.1080/00207540701644219

Zobolas, G. I., Tarantilis, C. D., & Ioannou, G. (2008). Exact, Heuristic and Meta-heuristic Algorithms for Solving Shop Scheduling Problems. In F. Xhafa & A. Abraham (Eds.), *Metaheuristics for Scheduling in Industrial and Manufacturing Applications* (pp. 1-31). Berlin, Germany: Springer.

Appendices

A. 3x3 Instance Data

In this appendix, data regarding the instance used to illustrate the UpdateSetupTimes procedure in Section 4.1.1 is presented. Figure 25 provides data regarding the setups: the three tables in the top of the figure are the setup matrices of each resource, and the table in the bottom identifies the setup characteristic of each operation.

Table 8 shows data regarding the eligible resources for each of the operations. The first two columns show the jobs and their operations. The following columns display the processing time (Proc. time), the standard capacity (Std. cap.) and the standard setup time (Std. ST) for each eligible resource in which operations can be performed. Note that all resources are finite capacity NWC.

For simplification purposes it is assumed that resources are always available and their capacity is always equal to 1. The release dates of operations take value zero (i.e. $r_{hj} = 0$) and the due dates of the jobs are the following: $d_1 = 9$, $d_2 = 12$, and $d_3 = 8$. Finally, there are no precedence constraints between jobs and the precedence constraints among operations of the same job are linear.

Resource: R1
Setup characteristic: ColourX

From \ To	White	Red
White	0 (Replace)	-1 (Add)
Red	1 (Add)	0 (Replace)

Resource: R2
Setup characteristic: ColourY

From \ To	Black	White
Black	0 (Replace)	1 (Add)
White	-1 (Add)	0 (Replace)

Resource: R3
Setup characteristic: Size

From \ To	010	020
010	0 (Replace)	1 (Add)
020	1 (Add)	0 (Replace)

Operation	**Setup characteristic**
$O_{1,1}$	Size (010)
$O_{2,1}$	ColourX (White)
$O_{1,2}$	Size (010)
$O_{2,2}$	ColourY (White)
$O_{3,2}$	ColourX (White)
$O_{1,3}$	ColourY(Black)
$O_{2,3}$	ColourX (Red)
$O_{3,3}$	Size (020)

Figure 25. Setup data of the 3 x 3 instance.

Table 8. Data regarding eligible resources for operations of the 3x3 instance.

		Resources								
		R1			R2			R3		
Job	Operation	Proc. time	Std. Cap.	Std. ST	Proc. time	Std. Cap.	Std. ST	Proc. time	Std. Cap.	Std. ST
1	O_1							2	1	1
	O_2	2	1	1						
2	O_1							2	1	1
	O_2				3	1	1			
	O_3	2	1	1						
3	O_1				2	1	1			
	O_2	2	1	1						
	O_3							2	1	1

B. Computational Results Data

Table 9 to Table 14 shows the results of each constructive method (JBJ, OBO, and RBR) with and without the improvement heuristics and local search. The first column of each table displays the instance name, followed by the overall best result (BR). Afterwards, for each algorithm, it is shown: the obtained results in terms of objective function (Result) and makespan (Mkp.); CPU times, in seconds; and gap (Gap_{BR}) between the algorithm result and the overall best known result. Additionally, in the last two rows of the tables, average and median values for makespan, CPU time and Gap_{BR} are provided.

Table 9. Results for the JBJ constructive method with and without the improvement heuristics and local search.

			JBJ				JBJ + Reassign				JBJ + External Exchange				JBJ + Internal Exchange			
	Instance	BR	Result	Mkp.	CPU (s)	Gap_{BR}	Result	Mkp.	CPU (s)	Gap_{BR}	Result	Mkp.	CPU (s)	Gap_{BR}	Result	Mkp.	CPU (s)	Gap_{BR}
1	Km01	4	4	67.60	0.12	0	4	67.60	0.06	0	4	67.60	0.11	0	4	67.60	0.11	0
2	Km02	7	10	177.10	0.13	3	10	177.10	0.08	3	10	177.10	0.08	3	10	177.10	0.05	3
3	Km03	7	10	104.50	0.14	3	8	104.50	0.12	1	8	73.20	0.11	1	10	104.50	0.08	3
4	Km04	5	14	136.40	0.13	9	11	135.00	0.25	6	14	136.40	0.06	9	14	136.40	0.07	9
5	Mk01	2	9	69.90	0.13	7	6	66.80	0.22	4	6	66.80	0.43	4	9	69.90	0.08	7
6	Mk02	1	6	119.60	0.13	5	5	91.30	0.18	4	5	122.00	0.18	4	6	119.60	0.08	5
7	Mk03	6	14	431.40	0.13	8	13	431.40	2.54	7	12	423.50	10.77	6	14	431.40	0.26	8
8	Mk04	4	12	97.50	0.15	8	11	103.20	0.52	7	8	112.50	1.09	4	11	97.10	0.13	7
9	Mk05	11	12	351.40	0.11	1	12	351.40	0.24	1	12	351.40	0.61	1	12	351.40	0.51	1
10	Mk06	2	8	216.70	0.15	6	8	216.70	1.45	6	7	226.40	13.72	5	8	216.70	0.26	6
11	Mk07	10	14	279.40	0.12	4	13	279.40	0.39	3	13	279.40	0.73	3	14	279.40	0.28	4
12	Mk08	7	16	780.00	0.15	9	15	799.80	6.10	8	15	799.80	13.38	8	15	782.40	2.36	8
13	Mk09	5	14	676.10	0.17	9	8	610.30	10.50	3	10	613.60	39.03	5	13	664.20	1.50	8
14	Mk10	6	17	595.90	0.21	11	13	580.50	12.01	7	11	587.20	47.06	5	14	579.70	3.66	8
15	Mk11	13	24	890.90	0.14	11	23	879.50	1.56	10	23	879.50	3.27	10	24	890.90	3.11	11
16	Mk12	16	24	853.50	0.18	8	21	875.50	2.68	5	20	821.60	5.96	4	23	839.00	1.29	7
17	Mk13	16	23	816.80	0.16	7	20	809.60	10.98	4	16	786.40	82.65	0	22	848.90	2.19	6
18	Mk14	18	27	1228.10	0.18	9	24	1195.10	16.35	6	24	1240.80	58.76	6	26	1244.00	3.97	8
19	Mk15	13	25	782.70	0.20	12	21	775.80	18.21	8	22	801.80	92.60	9	25	782.70	1.55	12
20	R01	6	48	1321.18	0.06	42	12	905.79	89.77	6	48	1321.18	266.97	42	40	1321.18	174.71	34
			Average	499.83	0.14	8.60		472.81	8.71	4.95		494.41	31.88	6.45		500.20	9.81	7.75
			Median	391.40	0.14	8.00		391.40	1.51	5.50		387.45	4.62	4.50		391.40	0.40	7.00

Table 10. Results for the JBJ constructive method with and without the improvement heuristics and local search (continuation).

			JBJ + Swap				JBJ + Reinsert-reassign				JBJ + Local search			
	Instance	BR	Result	Mkp.	CPU (s)	Gap_{BR}	Result	Mkp.	CPU (s)	Gap_{BR}	Result	Mkp.	CPU (s)	Gap_{BR}
1	Km01	4	4	67.60	0.11	0	4	67.60	0.13	0	4	67.60	0.15	0
2	Km02	7	10	177.10	0.08	3	9	181.50	0.27	2	10	177.10	0.13	3
3	Km03	7	10	104.50	0.10	3	8	104.50	0.32	1	8	104.50	0.21	1
4	Km04	5	13	136.00	0.20	8	11	135.00	1.02	6	11	135.00	0.40	6
5	Mk01	2	8	70.90	0.23	6	5	57.10	0.70	3	6	66.80	0.43	4
6	Mk02	1	6	119.60	0.14	5	5	91.30	0.50	4	5	91.30	0.33	4
7	Mk03	6	14	431.40	2.55	8	13	431.40	18.00	7	13	431.40	4.15	7
8	Mk04	4	11	97.10	0.89	7	10	91.30	2.96	6	11	97.10	0.62	7
9	Mk05	11	12	351.40	2.15	1	12	351.40	2.24	1	12	351.40	0.71	1
10	Mk06	2	8	216.70	2.65	6	8	216.70	30.62	6	8	216.70	1.72	6
11	Mk07	10	14	279.40	1.25	4	13	279.40	3.15	3	13	279.40	0.81	3
12	Mk08	7	15	788.30	15.37	8	15	799.80	33.17	8	14	772.90	12.30	7
13	Mk09	5	12	664.20	28.18	7	8	585.30	84.14	3	13	664.20	11.37	8
14	Mk10	6	15	566.60	16.38	9	12	580.40	125.47	6	13	587.40	22.89	7
15	Mk11	13	22	884.60	18.83	9	23	879.50	17.99	10	23	879.50	6.84	10
16	Mk12	16	22	829.30	24.28	6	20	858.70	55.16	4	21	875.50	6.60	5
17	Mk13	16	20	773.80	25.75	4	19	820.70	390.18	3	18	835.40	38.49	2
18	Mk14	18	26	1233.40	44.32	8	24	1195.10	357.61	6	24	1204.40	33.34	6
19	Mk15	13	24	794.20	31.43	11	21	775.80	312.25	8	21	775.80	28.80	8
20	R01	6	34	1315.51	3308.67	28	11	839.99	3613.24	5	13	884.71	283.00	7
			Average	495.08	176.18	7.05		467.12	252.46	4.60		474.91	22.66	5.10
			Median	391.40	2.60	6.50		391.40	18.00	4.50		391.40	2.94	6.00

Table 11. Results for the OBO constructive method with and without the improvement heuristics and local search.

			OBO				OBO + Reassign				OBO + External Exchange				OBO + Internal Exchange			
	Instance	BR	Result	Mkp.	CPU (s)	Gap_{BR}	Result	Mkp.	CPU (s)	Gap_{BR}	Result	Mkp.	CPU (s)	Gap_{BR}	Result	Mkp.	CPU (s)	Gap_{BR}
1	Km01	4	4	64.50	0.12	0	4	64.50	0.13	0	4	64.50	0.11	0	4	64.50	0.06	0
2	Km02	7	9	175.50	0.15	2	9	175.50	0.07	2	9	175.50	0.08	2	9	175.50	0.08	2
3	Km03	7	8	69.40	0.15	1	8	69.40	0.10	1	8	69.40	0.08	1	8	69.40	0.08	1
4	Km04	5	11	149.70	0.12	6	9	149.00	0.20	4	9	150.00	0.32	4	11	149.70	0.06	6
5	Mk01	2	9	49.90	0.12	7	7	49.90	0.20	5	8	71.70	0.15	6	9	49.90	0.08	7
6	Mk02	1	3	76.70	0.12	2	3	76.70	0.11	2	3	76.70	0.21	2	3	76.70	0.08	2
7	Mk03	6	11	241.20	0.18	5	9	241.10	2.41	3	8	236.50	6.14	2	11	241.20	0.22	5
8	Mk04	4	8	70.50	0.13	4	8	70.50	0.35	4	8	70.50	0.58	4	8	70.50	0.13	4
9	Mk05	11	14	347.90	0.13	3	12	334.30	0.42	1	12	334.30	0.55	1	12	334.30	0.53	1
10	Mk06	2	6	161.80	0.18	4	4	144.20	3.10	2	2	156.70	19.89	0	4	161.00	0.40	2
11	Mk07	10	13	279.40	0.12	3	13	279.40	0.24	3	13	279.40	0.59	3	12	279.40	0.27	2
12	Mk08	7	15	681.10	0.33	8	11	665.10	6.00	4	12	689.70	13.63	5	12	669.60	2.15	5
13	Mk09	5	13	568.10	0.42	8	10	561.80	6.98	5	7	539.10	37.15	2	11	585.50	2.31	6
14	Mk10	6	18	618.00	0.41	12	11	541.30	11.21	5	9	573.50	32.98	3	11	541.30	1.53	5
15	Mk11	13	23	887.40	0.23	10	23	887.40	1.03	10	23	887.40	1.82	10	23	887.40	2.37	10
16	Mk12	16	23	807.70	0.33	7	20	819.70	2.64	4	20	808.70	8.00	4	23	807.70	1.10	7
17	Mk13	16	21	641.90	0.43	5	17	674.70	10.17	1	17	690.00	88.23	1	17	677.90	2.09	1
18	Mk14	18	26	1161.00	0.55	8	24	1156.70	14.96	6	24	1156.70	33.16	6	25	1161.10	3.67	7
19	Mk15	13	20	597.10	0.62	7	18	618.70	19.83	5	15	584.50	58.56	2	20	597.10	1.54	7
20	R01	6	7	572.41	3.76	1	7	572.41	30.89	1	7	572.41	235.11	1	7	572.41	12.91	1
			Average	411.06	0.43	5.15		407.62	5.55	3.40		409.36	26.87	2.95		408.61	1.58	4.05
			Median	313.65	0.18	5.00		306.85	1.72	3.50		306.85	3.98	2.00		306.85	0.47	4.50

Table 12. Results for the OBO constructive methods with and without the improvement heuristics and local search (continuation).

			OBO + Swap				OBO + Reinsert-reassign				OBO + Local search			
	Instance	BR	Result	Mkp.	CPU (s)	Gap_{BR}	Result	Mkp.	CPU (s)	Gap_{BR}	Result	Mkp.	CPU (s)	Gap_{BR}
1	Km01	4	4	64.50	0.04	0	4	64.50	0.13	0	4	64.50	0.18	0
2	Km02	7	9	175.50	0.08	2	9	175.50	0.20	2	9	175.50	0.15	2
3	Km03	7	8	69.40	0.12	1	8	69.40	0.32	1	8	69.40	0.20	1
4	Km04	5	9	146.00	0.18	4	8	146.70	0.88	3	9	149.00	0.38	4
5	Mk01	2	6	51.00	0.28	4	7	49.90	0.55	5	7	49.90	0.39	5
6	Mk02	1	3	76.70	0.23	2	3	76.70	0.46	2	3	76.70	0.20	2
7	Mk03	6	11	241.20	2.47	5	9	242.70	16.35	3	9	241.10	4.13	3
8	Mk04	4	8	70.50	0.90	4	7	64.30	3.77	3	8	70.50	0.44	4
9	Mk05	11	12	334.30	2.23	1	12	334.30	2.34	1	12	334.30	1.37	1
10	Mk06	2	4	160.80	3.74	2	4	144.20	34.63	2	4	161.00	3.48	2
11	Mk07	10	11	277.30	2.07	1	12	272.50	3.69	2	12	279.40	0.83	2
12	Mk08	7	9	656.30	42.86	2	11	664.50	32.07	4	9	655.00	17.31	2
13	Mk09	5	9	582.10	20.52	4	10	561.80	61.46	5	6	580.10	18.95	1
14	Mk10	6	10	544.90	26.94	4	8	515.50	78.86	2	11	541.30	14.25	5
15	Mk11	13	23	887.40	14.13	10	23	887.40	19.14	10	23	887.40	2.88	10
16	Mk12	16	22	814.10	14.57	6	20	819.70	48.37	4	20	819.70	6.23	4
17	Mk13	16	18	677.30	37.56	2	18	691.00	311.83	2	17	677.90	17.81	1
18	Mk14	18	24	1156.70	46.69	6	24	1156.70	321.41	6	24	1156.70	32.46	6
19	Mk15	13	18	619.80	45.17	5	16	598.90	330.99	3	18	618.70	36.01	5
20	R01	6	7	572.41	292.18	1	7	572.41	3538.03	1	7	572.41	39.87	1
			Average	408.91	27.65	3.30		405.43	240.27	3.05		409.03	9.88	3.05
			Median	305.80	3.11	3.00		303.40	17.75	2.50		306.85	3.18	2.00

Table 13. Results for the RBR constructive method with and without the improvement heuristics and local search.

			RBR				RBR + Reassign				RBR + External Exchange				RBR + Internal Exchange			
	Instance	BR	Result	Mkp.	CPU (s)	Gap_{BR}	Result	Mkp.	CPU (s)	Gap_{BR}	Result	Mkp.	CPU (s)	Gap_{BR}	Result	Mkp.	CPU (s)	Gap_{BR}
1	Km01	4	4	67.90	0.13	0	4	67.90	0.12	0	4	67.90	0.14	0	4	67.90	0.14	0
2	Km02	7	10	190.80	0.13	3	10	190.80	0.08	3	10	190.80	0.10	3	10	190.80	0.06	3
3	Km03	7	10	100.60	0.15	3	8	94.00	0.15	1	8	94.00	0.08	1	10	100.60	0.06	3
4	Km04	5	13	165.00	0.12	8	10	165.00	0.27	5	9	164.00	0.59	4	13	165.00	0.07	8
5	Mk01	2	10	80.00	0.15	8	8	81.00	0.18	6	7	80.00	0.34	5	9	80.00	0.10	7
6	Mk02	1	7	82.80	0.12	6	5	82.00	0.22	4	4	80.80	0.30	3	7	82.80	0.08	6
7	Mk03	6	14	326.50	0.15	8	12	293.80	1.96	6	11	327.60	6.41	5	13	328.10	0.32	7
8	Mk04	4	14	143.50	0.15	10	12	100.40	0.42	8	10	143.20	1.38	6	14	143.50	0.12	10
9	Mk05	11	12	333.40	0.12	1	12	333.40	0.20	1	12	333.40	0.72	1	12	333.40	0.47	1
10	Mk06	2	10	204.90	0.15	8	6	191.80	2.88	4	6	177.00	7.36	4	8	202.50	0.38	6
11	Mk07	10	16	278.20	0.12	6	14	278.20	0.35	4	13	278.20	1.30	3	14	272.60	0.35	4
12	Mk08	7	18	728.60	0.20	11	12	714.30	5.37	5	14	712.30	25.18	7	16	731.10	4.10	9
13	Mk09	5	14	550.40	0.18	9	13	555.20	7.82	8	12	546.90	18.03	7	14	550.40	1.61	9
14	Mk10	6	14	549.40	0.20	8	11	540.70	9.80	5	10	547.50	40.74	4	13	541.80	3.32	7
15	Mk11	13	23	867.20	0.18	10	23	867.20	0.80	10	23	867.20	3.18	10	20	864.70	9.50	7
16	Mk12	16	25	815.40	0.21	9	22	818.60	2.91	6	21	850.00	8.16	5	23	815.40	3.63	7
17	Mk13	16	22	700.70	0.23	6	21	700.70	11.52	5	19	709.00	51.88	3	21	700.70	3.65	5
18	Mk14	18	28	1066.10	0.33	10	25	1059.30	11.77	7	25	1059.30	24.57	7	25	1066.60	10.85	7
19	Mk15	13	26	652.20	0.27	13	20	675.90	45.92	7	19	649.50	101.08	6	26	652.20	1.45	13
20	R01	6	11	716.24	1.85	5	9	744.66	59.35	3	8	689.58	378.05	2	11	716.24	17.10	5
			Average	430.99	0.26	7.10		427.74	8.10	4.90		428.41	33.48	4.30		430.32	2.87	6.20
			Median	329.95	0.15	8.00		313.60	1.38	5.00		330.50	4.80	4.00		330.75	0.43	7.00

Table 14. Results for the RBR constructive method with and without the improvement heuristics and local search (continuation).

			RBR + Swap				RBR + Reinsert-reassign				RBR + Local search			
	Instance	BR	Result	Mkp.	CPU (s)	Gap_{BR}	Result	Mkp.	CPU (s)	Gap_{BR}	Result	Mkp.	CPU (s)	Gap_{BR}
1	Km01	4	4	67.90	0.05	0	4	67.90	0.12	0	4	67.90	0.13	0
2	Km02	7	10	190.80	0.10	3	10	190.80	0.30	3	10	190.80	0.18	3
3	Km03	7	10	100.60	0.11	3	8	94.00	0.23	1	8	94.00	0.20	1
4	Km04	5	12	165.00	0.34	7	10	165.00	1.45	5	10	165.00	0.43	5
5	Mk01	2	8	80.00	0.33	6	8	81.00	0.73	6	7	70.60	0.37	5
6	Mk02	1	6	82.80	0.32	5	5	82.00	0.62	4	5	82.00	0.37	4
7	Mk03	6	12	328.50	7.28	6	11	330.00	14.20	5	13	328.10	2.35	7
8	Mk04	4	14	143.50	1.17	10	12	100.40	3.36	8	12	100.40	0.70	8
9	Mk05	11	12	333.40	2.96	1	12	333.40	1.03	1	12	333.40	0.74	1
10	Mk06	2	8	204.90	9.87	6	6	189.20	30.82	4	6	177.20	4.39	4
11	Mk07	10	14	272.60	2.01	4	13	272.60	3.35	3	13	272.60	0.93	3
12	Mk08	7	17	734.60	53.58	10	12	714.10	44.15	5	11	716.80	11.34	4
13	Mk09	5	12	588.10	36.68	7	13	555.20	84.47	8	13	555.20	14.50	8
14	Mk10	6	11	531.00	74.72	5	11	540.70	129.83	5	11	534.00	19.65	5
15	Mk11	13	18	875.40	93.80	5	23	867.20	14.15	10	20	864.70	15.28	7
16	Mk12	16	21	816.80	64.82	5	22	818.60	49.93	6	23	815.40	7.91	7
17	Mk13	16	21	698.20	51.93	5	21	700.70	283.35	5	20	726.50	18.00	4
18	Mk14	18	22	1022.20	234.87	4	25	1059.30	222.41	7	25	1066.60	32.26	7
19	Mk15	13	22	656.50	71.90	9	21	660.80	365.72	8	19	680.60	79.28	6
20	R01	6	9	695.74	568.45	3	8	696.49	3814.80	2	9	744.66	129.98	3
			Average	429.43	63.76	5.20		425.97	253.25	4.80		429.32	16.95	4.60
			Median	330.95	8.58	5.00		331.70	14.18	5.00		330.75	3.37	4.50